ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

DESARROLLO DE COMPOSICIONES DE SERVICIOS WEB EN WS-BPEL 2.0 PARA LA APLICACIÓN DE LA PRUEBA DE MUTACIONES

Álvaro Cortijo García

13 de septiembre de 2012

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

DESARROLLO DE COMPOSICIONES DE SERVICIOS WEB EN WS-BPEL 2.0 PARA LA APLICACIÓN DE LA PRUEBA DE MUTACIONES

- Departamento: Departamento de Ingeniería Informática
- Director del proyecto: Antonia Estero Botaro
- Autor del proyecto: Álvaro Cortijo García

Cádiz, 13 de septiembre de 2012

Fdo.: Álvaro Cortijo García

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Motivación y contexto

En este capítulo del documento se describe la motivación del proyecto a través de una introducción, así como, los objetivos que se tratan de satisfacer en este Proyecto Fin de Carrera (PFC). A lo largo de este capítulo también se detallan una serie de conceptos básicos, como la prueba de mutaciones y las distintas tecnologías que han sido necesarias para el desarrollo del proyecto.

1.1. Introducción

Este PFC se ha realizado en colaboración con el grupo de investigación UCASE, el cual pertenece al Departamento de Ingeniería Informática de la Universidad de Cádiz.

El grupo de investigación UCASE, al que pertenece mi tutora, Antonia Estero Botaro, me propuso la realización de este PFC. Situación que me pareció de gran interés puesto que me permitiría adquirir conocimientos de gran utilidad para mí, como la prueba de mutaciones, el lenguaje WS-BPEL 2.0 (Web Services Business Process Execution Language) o Servicios Web, entre otros.

Este PFC está enmarcado dentro de una línea de trabajo del grupo de investigación UCASE que se centra en la aplicación de la técnica de prueba de mutaciones a WS-BPEL 2.0. El lenguaje WS-BPEL 2.0 permite crear procesos de negocio a partir de servicios web preexistentes, facilitando el desarrollo de aplicaciones distribuidas, de manera que se obtienen a un coste menor y en un tiempo menor. Debido a la creciente importancia que los servicios web están adquiriendo, y a su progresivo impacto en la economía, la prueba de este tipo de software requiere de un especial interés. El grupo de investigación UCASE trabaja desde hace varios años en una línea de investigación que consiste en la aplicación de la técnica conocida como prueba de mutaciones a composiciones WS-BPEL. En este sentido, han definido e implementado un conjunto de operadores de mutación para WS-BPEL 2.0 que modela los errores que suelen cometer los programadores [1], así como un conjunto de operadores de cobertura [2]. Todos estos operadores están integrados en la herramienta MuBPEL [3], que permite aplicar de forma automática la prueba de mutaciones a este tipo de composiciones.

Para determinar si todos los operadores de mutación definidos son válidos es imprescindible evaluarlos. Para poder realizar esta evaluación son necesarias composiciones WS-BPEL que permitan ejercitarlos. Sin embargo, no existen muchas composiciones WS-BPEL de libre disposición que permitan aplicar la totalidad de los operadores definidos, por lo que se necesitan composiciones adicionales especialmente adaptadas a la aplicación de estos operadores.

1.2. Objetivos

El objetivo principal que se persigue con la realización de este PFC es el desarrollo de composiciones de servicios web en WS-BPEL 2.0 especialmente adaptadas a la aplicación de los operadores de

mutación definidos para WS-BPEL 2.0. El propósito a alcanzar con estas composiciones es permitir la evaluación de todos los operadores de mutación de WS-BPEL 2.0. Se parte de una situación en la que se dispone de composiciones que permiten evaluar un subconjunto de los operadores definidos, por lo que las composiciones a desarrollar deben complementar a las ya existentes en cuanto a la evaluación del resto de los operadores de mutación.

Para conseguir el objetivo principal hemos subdividido éste en los siguientes:

- Elección del editor gráfico a utilizar para implementar las composiciones. Aunque el código WS-BPEL se puede escribir directamente mediante un editor de texto, se pueden utilizar distintas herramientas gráficas que facilitan la realización de las composiciones WS-BPEL. Estas herramientas también ofrecen facilidades para escribir los ficheros complementarios que deben acompañar a la composición.
- Desarrollo de cada una de las composiciones, teniendo en cuenta que estas deben ir acompañadas de un conjunto de ficheros complementarios.
  - La composición se implementa en un fichero con extensión *bpel* que define el proceso de negocio.
  - El proceso BPEL se comunica con distintos servicios, la especificación de estos servicios se realiza en un fichero con extensión *wsdl*.
  - Para poder ejecutar la composición WS-BPEL se necesita disponer de un conjunto de casos de prueba. Los ficheros de casos de prueba tienen un formato especial basado en XML y tienen extensión *bpts*.
  - Dado que el formato del fichero de casos de prueba es complejo, se ha conseguido simplificarlo mediante los ficheros de casos de prueba basados en plantillas, de esta forma se separa la estructura del fichero de casos de prueba de los datos concretos que lo componen.
  - Un fichero con extensión *spec* que permita generar de forma aleatoria los datos de prueba para una determinada composición.
- Las composiciones realizadas deben integrarse en un repositorio de composiciones que mantiene el grupo de investigación UCASE. Debiendo cumplirse una serie de requisitos para que las composiciones puedan formar parte de dicho repositorio.

Para la realización de este proyecto ha sido necesario aprender el lenguaje WS-BPEL 2.0. Así como WSDL para la descripción de los servicios web, BPELUnit para la realización de las pruebas unitarias en las composiciones, Velocity para la realización de las pruebas unitarias basadas en plantillas, XMLSchema, SOAP y XPath.

1.3. Conceptos básicos

Esta sección describe la técnica de la prueba de mutaciones y las tecnologías que se han manejado para la realización de este PFC.

1.3.1. La prueba de mutaciones

La *prueba de mutaciones* [4] es una técnica de prueba del software basada en fallos. La prueba de mutaciones consiste en introducir pequeños cambios sintácticos en el programa a probar mediante la aplicación de los denominados *operadores de mutación*. Los operadores de mutación suelen modelar los principales fallos que suelen cometer los programadores al implementar un programa o bien, pueden

permitir aplicar criterios de cobertura. Los nuevos programas generados mediante la aplicación de los operadores de mutación se denominan *mutantes*. La prueba de mutaciones permite conseguir dos objetivos: probar el programa y medir la calidad del conjunto de casos de pruebas que se está utilizando, permitiendo además mejorar su calidad si no se considera adecuada.

A continuación, podemos observar un ejemplo de aplicación de un operador de mutación. El código original es una expresión lógica en la que comprobamos si la variable *stock* es mayor que una determinada cantidad.

```
1 $stock > 100
```

El mutante generado, a partir de ese código, por un operador de mutación podría ser el de cambiar el operador lógico por otro, y en vez de comprobar si la variable *stock* es mayor que una determinada cantidad, comprobar si es menor.

```
1 $stock < 100
```

Una vez que se han obtenido los distintos mutantes de un determinado programa, se ejecutan contra un conjunto de casos de prueba y se comparan los resultados obtenidos por los mutantes con los del programa original. Si la salida del mutante coincide con la del programa original, decimos que el mutante ha quedado vivo. Por contra, si ambas salidas no coinciden decimos que el mutante ha muerto. También existe la posibilidad de que un mutante no pueda ser ejecutado, en este caso decimos que el mutante es no válido.

Un mutante puede quedar vivo por dos razones:

- No disponemos del caso de prueba adecuado que lo detecte, decimos que se trata de un mutante persistente.
- Se trata de un mutante equivalente, es decir, un mutante que siempre va a producir la misma salida que el programa original.

En el primer caso, añadiendo nuevos casos de prueba a nuestro conjunto podremos matar al mutante, aumentando así la calidad del conjunto de casos de prueba. Cuando un conjunto de casos de prueba mata a todos los mutantes no equivalentes producidos por un programa se dice que es un conjunto de casos de prueba *adecuado*.

La calidad de un conjunto de casos de prueba se puede calcular mediante el cociente entre el número de mutantes muertos y el número de mutantes no equivalentes.

1.3.2. El lenguaje WS-BPEL 2.0

El lenguaje WS-BPEL 2.0 (Web Services Business Process Execution Language) es un lenguaje basado en XML estandarizado por OASIS [5] con el que podemos especificar el comportamiento de un proceso de negocio, basado en la interacción con servicios web. Estos servicios se especifican mediante el lenguaje WSDL (Web Services Description Language), a través de este lenguaje se pueden definir los mensajes mediante los que pueden interactuar los distintos servicios web durante su comunicación, así como, la forma de establecer la comunicación entre los distintos servicios web y los protocolos de comunicación usados.

La estructura de un proceso WS-BPEL se divide en las siguientes cuatro secciones:

1. Definición de las relaciones con los socios externos, es decir, el cliente que utiliza el proceso de negocio y los WS a los que llama el proceso.

2. Definición de las variables que emplea el proceso, mediante Web Services Description Language (WSDL) y XML Schema (XSD).

3. Definición de los distintos tipos de manejadores que puede utilizar el proceso. Los distintos manejadores disponibles en WS-BPEL son:

   - **Manejadores de fallos:** A través de los elementos *catch* permiten determinar las actividades a llevar a cabo en caso de que se produzca un fallo específico, y mediante el elemento *catchall* las actividades relacionadas con cualquier otro fallo no especificado anteriormente.
   - **Manejadores de compensación:** Permite deshacer el trabajo que ya ha sido realizado cuando se produce un fallo en la ejecución de un proceso.
   - **Manejadores de terminación:** Permite especificar las actividades a llevar a cabo si se debe terminar la ejecución de un *scope*.
   - **Manejadores de eventos:** A través de los elementos *onEvent* determinan las acciones a realizar cuando se produce un evento dado, y mediante un elemento *onAlarm* las actividades a realizar si después de un tiempo determinado no se ha producido ningún evento.

4. Descripción del comportamiento del proceso de negocio a través de las actividades que proporciona el lenguaje.

Todos los elementos anteriores son globales por omisión. Aunque se tiene la posibilidad de declararlos de forma local utilizando el contenedor *scope*, con el que se puede dividir el proceso de negocio en diferentes ámbitos.

Los principales elementos que constituyen un proceso WS-BPEL son las actividades, que pueden tener atributos asociados y una serie de contenedores, que también pueden tener atributos asociados e incorporar distintos elementos. Las actividades pueden ser de dos tipos: básicas y estructuradas.

- **Actividades básicas:** Estas actividades tienen una labor específica en el proceso de negocio (recepción de mensajes, invocación de servicios, manipulación de datos, etc.).
- **Actividades estructuradas:** Estas actividades están compuestas por otras actividades y son las que definen la lógica de negocio.

Las distintas actividades que nos podemos encontrar en el lenguaje WS-BPEL 2.0 son las siguientes:

- **<receive>:** Mediante esta actividad se permite al proceso de negocio esperar para la llegada de un mensaje entrante.
- **<reply>:** Esta actividad permite que el proceso de negocio pueda enviar un mensaje en respuesta a uno entrante.
- **<invoke>:** Esta actividad permite al proceso de negocio invocar a una operación ofrecida por un socio.
- **<assign>:** Actividad utilizada para actualizar los valores de las variables.
- **<throw>:** Esta actividad se utiliza para generar un fallo desde el proceso de negocio.
- **<exit>:** Esta actividad se utiliza para salir inmediatamente del proceso de negocio.
- **<wait>:** Actividad utilizada para esperar un período de tiempo dado o hasta que se alcance un cierto punto en el tiempo.

- **<empty>:** Se trata de una actividad mediante la que no se realiza nada, pero puede ser útil para la sincronización de actividades concurrentes.
- **<sequence>:** Actividad que se utiliza para definir un conjunto de actividades que se realizarán de forma secuencial en un orden léxico.
- **<if>:** Mediante esta actividad es posible seleccionar una determinada actividad a ejecutar entre un conjunto de opciones.
- **<while>:** Proporciona la ejecución repetida de la actividad hija que contiene mientras que la condición especificada sea verdadera.
- **<repeatUntil>:** Proporciona la ejecución repetida de la actividad hija que contiene hasta que la condición especificada sea verdadera.
- **<forEach>:** Proporciona la ejecución repetida de la actividad <*scope*> hija que contiene exactamente N+1 veces, donde N es el valor correspondiente a <*finalCounterValue*> menos el valor de <*startCounterValue*>. Esta actividad consta del atributo *parallel*, que si toma el valor "yes", se ejecutarían paralelamente las N+1 instancias del <*scope*> contenido.
- **<pick>:** Se trata de una actividad mediante la que, a través de los elementos <*onMessage*> permite determinar las actividades a realizar cuando se recibe un determinado mensaje y, a través de un elemento <*onAlarm*>, especificar las actividades a realizar cuando se alcance un determinado período de tiempo.
- **<flow>:** Esta actividad se utiliza para proporcionar concurrencia mediante la ejecución de todas sus actividades hijas en paralelo. También es posible definir dependencias de control sobre las actividades que contiene mediante los enlaces de sincronización.
- **<scope>:** Actividad que puede contener una actividad hija y permite la definición de forma local de los socios de negocio, variables y manejadores. Esta actividad tiene el atributo *isolated*, cuando dicho atributo toma el valor "yes" se protege el acceso a las variables compartidas, evitando que otro <*scope*> que utilice las mismas variables pueda acceder a ellas hasta que no haya terminado la ejecución del primer <*scope*>.
- **<compensate>:** Actividad utilizada para iniciar una compensación de todos los <*scope*> internos que ya han sido completados con éxito.
- **<compensateScope>:** Actividad utilizada para iniciar una compensación de un <*scope*> específico interno que ya ha sido completado con éxito.
- **<rethrow>:** Actividad utilizada para volver a lanzar un fallo que ha sido capturado por un manejador de fallos.
- **<validate>:** Esta actividad se utiliza para validar los valores de las variables con respecto a su definición de datos WSDL y XML asociada.
- **<extensionActivity>:** Esta actividad se utiliza para extender WS-BPEL mediante la introducción de un nuevo tipo de actividad.

A continuación, mostramos un ejemplo de código en lenguaje WS-BPEL 2.0:

```
<flow> ← Actividad estructurada
  <links> ← Contenedor
    <link name=
        "comprobarVuelo-A-reservarVuelo" ← Atributo/> ← Elemento
  </links>
  <invoke name="comprobarVuelo" . . . > ← Actividad básica
    <sources> ← Contenedor
      <source linkName=
          "comprobarVuelo-A-reservarVuelo" ← Atributo/> ← Elemento
    </sources>
  </invoke>
  <invoke name="comprobarHotel" . . . />
  <invoke name="comprobarAlquilerCoche" . . . />
  <invoke name="reservarVuelo" . . . >
    <targets> ← Contenedor
      <target linkName=
          "comprobarVuelo-A-reservarVuelo" /> ← Elemento
    </targets>
  </invoke>
</flow>
```

En el código anterior tenemos una actividad estructurada *flow* que contiene un conjunto de actividades *invoke* (*comprobarVuelo*, *comprobarHotel*, *comprobarAlquilerCoche* y *reservarVuelo*) que se ejecutarían en paralelo cuando la actividad *flow* comenzara. No obstante, en determinadas ocasiones puede que sea necesario establecer una sincronización entre algunas de estas actividades que están ejecutándose concurrentemente. En este caso, sólo tendría sentido reservar un vuelo una vez que se haya comprobado que dicho vuelo es correcto. Por lo tanto, es necesario establecer un enlace o *link* entre ambas actividades, introduciendo así, una dependencia entre ambas actividades que obligará a la actividad objetivo del enlace a ejecutarse solo cuando se haya completado la actividad fuente del enlace.

En WS-BPEL pueden aplicarse distintos lenguajes de expresiones. Todos los motores WS-BPEL estándar soportan el lenguaje XML Path Language 1.1, se trata de un lenguaje declarativo con el que se puede realizar consultas sobre documentos XML y dispone de operadores aritméticos, relacionales y lógicos con una sintaxis similar a la de los lenguajes tradicionales.

1.3.3. WSDL

WSDL 2.0 (Web Services Description Language) [6] es un lenguaje basado en XML utilizado para la descripción de servicios web. WSDL separa la descripción abstracta de un servicio, con la que se establece en términos generales la funcionalidad del servicio, de la descripción específica del mismo, que determina la ubicación del servicio y cómo llamarlo.

En la figura 1.1 podemos observar el formato de un documento WSDL.

En el nivel abstracto, los servicios web se describen mediante los mensajes que intercambia, cuya estructura se define mediante los elementos *type*, y las operaciones, que asocian uno o más mensajes con un patrón de intercambio de mensajes con el que identificar la secuencia y cardinalidad de los mensajes intercambiados. Estas operaciones se agrupan bajo una determinada interfaz, denominada *portType*.

Los tipos de mensajes que puede haber en una operación son: *input*, que define un mensaje de entrada, *output*, que define un mensaje de salida, y *fault*, que define un mensaje de excepción. Además, se pueden distinguir cuatro tipos de operaciones, que se definen por los mensajes que pueden enviar o recibir:

- **One-Way:** Se recibe un mensaje del cliente del servicio.

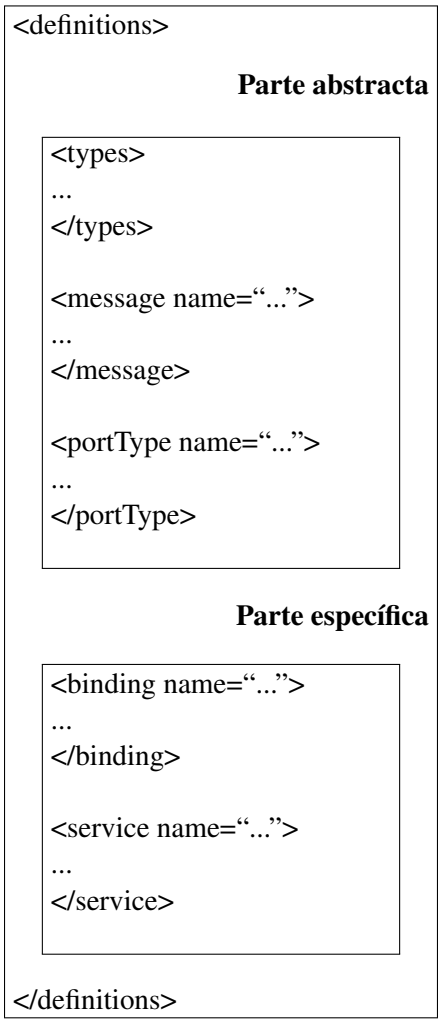

Figura 1.1: Formato de un documento WSDL

- **Request-Response:** Se recibe un mensaje del cliente del servicio y se le envía un mensaje de respuesta.
- **Solicit-Response:** El servicio envía un mensaje y se queda a la espera de una respuesta.
- **Notification:** El servicio envía un mensaje y no espera ninguna respuesta.

En el nivel específico, mediante un *binding* se especifican el formato de intercambio de información y el protocolo de transporte para una o más interfaces. Mediante los *binding* se definen los puertos (*ports*), a través de los cuales se integra la especificación de un *binding* con la dirección del servicio. Estos puertos se agrupan dando lugar a un servicio (*service*), tratándose como la representación completa de un servicio web.

Además, en los ficheros *wsdl* se pueden definir los distintos *partnerLinkType* necesarios para la interacción con los distintos servicios, así como, las propiedades necesarias para poder establecer una correlación entre mensajes.

Se ha combinado WSDL y SOAP, que es uno de los protocolos más utilizados en WSDL para los elementos *binding*. Cuándo un cliente se conecta a un servicio web, puede determinar qué funciones están disponibles a través del WSDL, y mediante la utilización de SOAP puede llamar a una de las funciones disponibles en el WSDL.

1.3.4. SOAP

SOAP (Simple Object Access Protocol) [7] es un protocolo diseñado para el intercambio de información estructurada. Se trata de un protocolo simple y extensible que está basado en XML y se utiliza para el intercambio de mensajes en un entorno distribuido. Un mensaje SOAP consiste de una envoltura SOAP (*envelope*), que contiene una cabecera SOAP (header) opcional, que puede contener información adicional, y un cuerpo SOAP (*body*) obligatorio, que contiene la información principal del mensaje.

Las capacidades básicas que incluye SOAP son las siguientes:

- **Formato de mensaje estandarizado:** SOAP especifica cómo la información se empaqueta en un documento XML estandarizado (un mensaje SOAP) para que se transfiera en una comunicación.
- **Convenciones de correspondencia:** La especificación SOAP comprende un conjunto de convenciones para mapear datos de aplicaciones en los mensajes SOAP, por ejemplo, especificando una llamada de procedimiento remoto.
- **Modelo de procesamiento:** Este modelo de procesamiento define los roles de los remitentes y receptores de mensajes SOAP, especificando qué partes de un mensaje deben ser procesadas por un rol.
- **Enlaces de protocolo de red:** SOAP especifica enlaces, que describen cómo los mensajes SOAP van a ser transportados sobre HTTP, SMTP y otros protocolos de transporte.

Mediante los binding SOAP podemos definir como un mensaje es creado a partir de un mensaje de entrada, salida o excepción de una operación de un WSDL. Podemos especificar como encuadrar la información dentro de un cuerpo SOAP mediante la definición de dos parámetros aplicados a un elemento *binding*, se trata de los atributos *style* y *use*, que corresponden directamente con el estilo y codificación SOAP.

El atributo *style* puede tomar los siguientes valores:

- **Document:** En este estilo las partes del mensaje se especifican como elementos, que contienen la información completa para un esquema XML definido.

- **RPC:** Las partes del mensaje son parámetros y valores de retorno. Normalmente, en este estilo las partes se especifican mediante tipos.

Mientras que en el atributo *use* podemos encontrar los siguientes valores:

- **Literal:** El formato de los datos sigue literalmente el especificado en el esquema XML.
- **Encoded:** El formato de los datos se codifica mediante un conjunto de reglas detalladas en la especificación SOAP.

A continuación podemos ver un ejemplo de como definir un *binding*:

```
<wsdl:binding name="ManagementBinding" type="tns:ManagementPT">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="ManagementOperation">
    <soap:operation
     soapAction="http://www.example.org/tradeIncome/ManagementOperation"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="managementfault">
      <soap:fault use="literal" name="managementfault"/>
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
```

1.3.5. XML Schema

XML Schema [8] es un lenguaje con el que se describe de forma precisa la estructura y las restricciones de un documento XML y los tipos de datos válidos para cada elemento y atributo. De forma que se pueden definir tipos de datos mediante tipos ya predefinidos y la agrupación de elementos y atributos de una manera determinada.

Este lenguaje es más complejo que las DTD (Document Type Definition) y se concibe como una alternativa a ellas. Una DTD es una descripción de la estructura y sintaxis de un documento XML. Con XML Schema se trata de ofrecer una mayor capacidad expresiva que con las DTD, proporcionando así una percepción del tipo de documento XML con un gran nivel de abstracción.

En XML Schema los elementos se clasifican dependiendo de los elementos y atributos que contienen. Los tipos de elementos pueden ser:

- **Simples:** elementos que no tienen elementos hijos ni atributos.
- **Complejos:** elementos que tienen elementos hijos y/o atributos.

A continuación podemos ver un ejemplo de como se describe la estructura de un elemento complejo con XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
3       <xsd:element name="cancion">
4           <xsd:complexType>
5              <xsd:sequence>
6                   <xsd:element name="titulo" type="xsd:string"/>
7                   <xsd:element name="cantante" type="xsd:string"/>
8              </xsd:sequence>
9              <xsd:attribute name="lanzamiento" type="xsd:date"/>
10          </xsd:complexType>
11     </xsd:element>
12 </xsd:schema>
```

1.3.6. XPath 2.0

El lenguaje XPath (XML Path Language) [9] es un lenguaje declarativo que permite la realización de consultas en documentos XML. Con este lenguaje se permite identificar fácilmente conjuntos de nodos en un documento XML mediante la creación de expresiones que recorren y procesan el documento XML.

Este lenguaje consta de operadores aritméticos, lógicos y relacionales con una sintaxis muy similar a la de los lenguajes tradicionales. XPath se basa en la representación arbórea del documento XML para seleccionar las partes deseadas del documento, pudiéndose filtrar nodos en función de su nombre, tipo, atributos, etc.

Un ejemplo de una expresión XPath puede ser el siguiente. En el que se seleccionan los *nodoC* cuyo atributo *name* tenga el valor *elegido* que sean hijos de un *nodoB* y que a su vez sean hijos de un *nodoA*.

```
1 /nodoA/nodoB/nodoC[@name="elegido"]
```

1.3.7. BPELUnit

BPELUnit [10] es un marco de pruebas unitarias automáticas, repetibles y de caja blanca para probar composiciones WS-BPEL. Las pruebas se realizan mediante la especificación de unos determinados casos de prueba, que consisten en una secuencia de operaciones de entrada y salida que son ejecutadas por el marco BPELUnit en nombre del cliente y de todos los socios del proceso BPEL. Estos casos de prueba se definen en un fichero con extensión *bpts* (BPELUnit Test Specification).

Los ficheros con extensión *bpts* son ficheros XML con los que podemos especificar qué datos serán enviados y qué datos esperan ser recibidos por el cliente (*clientTrack*) y cada socio del proceso (*partnerTrack*), no siendo necesario que todos los socios interactúen en un determinado caso de prueba.

El elemento raíz de un fichero con extensión *bpts* se denomina *testsuite* y consta los siguientes elementos:

- **name:** Nombre con el se identifica al conjunto de pruebas.
- **baseURL:** BPELUnit utiliza una URL local para simular los socios del proceso. La URL de cada socio es la concatenación de la URL base más el nombre de cada socio. La URL base predeterminada es http://localhost:7777/ws.
- **deployment:** Contiene la información relativa al despliegue del proceso y consta de:
  - **put:** Detalla el proceso a desplegar mediante su nombre, el motor con el que se va a ejecutar, el fichero WSDL con la especificación del servicio y el fichero bpr que contiene el servicio.

- **partner:** Mediante este elemento se especifican los socios del proceso, que pueden ser varios o ninguno. Cada socio se especifica mediante un nombre, con el que se identifica, y mediante el fichero WSDL que contenga su descripción.

- **testcases:** Elemento que contiene los distintos casos de prueba (*testcase*).

  Cada caso de prueba se identifica mediante un nombre. Además, un caso de prueba puede estar basado en otro definido anteriormente mediante el atributo *basedOn* y mediante el atributo *vary* se puede determinar si el caso de prueba se debe ejecutar varias veces. Por otro lado, mediante el atributo *abstract* se consigue que el caso de prueba no pueda ejecutarse al ser un caso de prueba abstracto.

  El contenido de los *clientTrack* y los *partnerTrack* consiste en una secuencia de actividades, que pueden ser las siguientes:

  - **sendOnly:** Mediante esta actividad se envía un único mensaje.
  - **receiveOnly:** Esta actividad espera un único mensaje y lo verifica.
  - **sendReceive:** Actividad con la que se envía un único mensaje, espera una respuesta síncrona y la verifica.
  - **receiveSend:** Esta actividad espera un único mensaje, lo verifica y envía una respuesta síncrona.
  - **sendReceiveAsync:** Mediante esta actividad se envía un único mensaje, se espera una respuesta asíncrona y se verifica.
  - **receiveSendAsync:** Actividad con la que se espera un único mensaje, lo verifica y envía una respuesta asíncrona.

En el siguiente fragmento de código podemos ver un ejemplo:

```
<tes:testSuite
   xmlns:ap="http://j2ee.netbeans.org/wsdl/squaresSum"
   xmlns:tns="http://xml.netbeans.org/schema/squaresSum"
   xmlns:tes="http://www.bpelunit.org/schema/testSuite">

  <tes:name>squaresSum</tes:name>
  <tes:baseURL>http://localhost:7777/ws</tes:baseURL>

  <tes:deployment>
    <tes:put name="squaresSum" type="activebpel">
      <tes:wsdl>squaresSum.wsdl</tes:wsdl>
      <tes:property name="BPRFile">squaresSum.bpr</tes:property>
    </tes:put>
  </tes:deployment>

  <tes:testCases>
    <tes:testCase name="ZeroIterations" basedOn="" abstract="false" vary="
       false">
      <tes:clientTrack>
        <tes:sendReceive
            service="ap:squaresSumService"
            port="squaresSumPort"
            operation="squaresSumOperation">

          <tes:send fault="false">
```

```
25              <tes:data>
26                <tns:squaresSumRequest>
27                  <tns:n>0</tns:n>
28                </tns:squaresSumRequest>
29              </tes:data>
30            </tes:send>
31
32            <tes:receive fault="false">
33              <tes:condition>
34                <tes:expression>tns:squaresSumResponse/tns:sum</
                      tes:expression>
35                <tes:value>0</tes:value>
36              </tes:condition>
37            </tes:receive>
38          </tes:sendReceive>
39        </tes:clientTrack>
40      </tes:testCase>
41    </tes:testCases>
42  </tes:testSuite>
```

Además, para la creación de los distintos casos de prueba, se puede utilizar el lenguaje de plantillas Apache Velocity, con el que se puede automatizar la generación de casos de pruebas.

1.3.8. Velocity

Velocity [11] consiste en un motor de plantillas basado en Java, que puede emplearse como una aplicación independiente con la que producir código fuente. Velocity puede ser utilizado para la creación de páginas web, SQL, PostScript y cualquier otro tipo de salida de plantillas. Fue creado para proporcionar una manera más simple, limpia y fácil de agregar contenido dinámico en una página web. La inclusión de dicho contenido dinámico se realiza mediante referencias, siendo una variable un tipo referencia.

Mediante la utilización de plantillas Velocity pueden generarse casos de pruebas automáticamente que contengan las mismas actividades, pero siendo distinto el contenido de los mensajes en cada caso de prueba. El contenido de los distintos mensajes puede formarse a través de la utilización de ciertas variables, pudiendo obtenerse del mismo fichero, o de un fichero externo, los datos para las variables en cada caso de prueba.

A continuación, veamos un ejemplo de un enunciado del lenguaje de plantillas Velocity:

```
1  #set ($par = "false")
```

Todos los enunciados comienzan con el carácter *#*, que contiene una directiva, que se utilizan para realizar una determinada acción. En el caso del ejemplo se trata de la directiva *set*, que sirve para establecer el valor de una referencia, que comienzan con el carácter *$*.

1.3.9. TestSpec

TestSpec [12] es un lenguaje con el que se describe el formato de los datos que pueden utilizarse en los casos de pruebas basados en plantillas Velocity. Los ficheros en los que se escribe este tipo de lenguaje tienen extensión *spec*.

La especificación de TestSpec está dividida en dos bloques: el primer bloque consta de las definiciones de tipos con sus restricciones (*typedef*), mientras que el segundo bloque consta de las declaraciones de tales tipos (*declaration*).

La sintaxis de la definición de un tipo es la siguiente:

```
typedef TipoBase (restriccion1 = valor1, restriccion2 = valor2...)
   NombreDelTipo;
```

Mientras que la declaración de un tipo se realiza de la siguiente manera:

```
Type Nombre;
```

Mediante la herramienta *TestGenerator* desarrollada dentro del grupo de investigación UCASE, a través de los ficheros *spec*, se generan unos ficheros con extensión *vm*. Estos ficheros contienen los datos que se utilizan para los casos de prueba basados en plantillas.

A continuación podemos ver un ejemplo de código de un fichero *spec*:

```
typedef int (min = 0, max = 500) Numero;

Numero n;
```

Y el contenido del fichero *vm* generado con *TestGenerator* a partir del fichero *spec*:

```
#set($n = [129, 67, 270])
```

Que se utilizará en los casos de pruebas basados en plantillas Velocity.

Capítulo 2

Planificación

En este capítulo del documento se detallan las distintas fases en las que se ha llevado a cabo el desarrollo de este PFC, así como la metodología que ha sido utilizada durante su desarrollo.

La elaboración de este proyecto se ha llevado a cabo desde julio de 2011, cuando inicié la toma de contacto con el grupo de investigación UCASE y tomó forma el propósito que se quería conseguir con este proyecto, hasta septiembre de 2012.

2.1. Metodología

La metodología que se ha seguido durante el desarrollo de este proyecto ha sido la metodología iterativa basada en prototipos. Esta metodología está especialmente planeada para cuando los requisitos no están del todo definidos al principio del proyecto.

En este proyecto, se realizan los prototipos de la composición a medida que se centra su desarrollo en la aplicación de determinados operadores de mutación. Mediante los prototipos se pueden implementar subconjuntos de las funciones que requiere el sistema. De esta manera, a cada prototipo se le va añadiendo la estructura necesaria para que se puedan aplicar los nuevos operadores en los que se centra su desarrollo.

2.2. Etapas del proyecto

2.2.1. Elicitación de requisitos

En esta etapa, se inició la toma de contacto con el grupo de investigación UCASE y, además, se me presentó la línea de investigación en la que trabaja el grupo relacionada con la prueba de mutaciones.

Posteriormente, se me indicó la necesidad de desarrollar nuevas composiciones WS-BPEL en las que se pudiesen aplicar ciertos operadores de mutación definidos para tal lenguaje. Mediante reuniones mantenidas con el grupo de investigación y la tutora de mi proyecto, Antonia Estero Botaro, se fueron analizado los requisitos necesarios de este PFC. Aunque, posteriormente, se fueron incrementando los requisitos a lo largo del proyecto.

2.2.2. Aprendizaje de nuevas tecnologías

En esta fase se realizó el aprendizaje de las distintas tecnologías necesarias para el desarrollo del proyecto. Especialmente, del lenguaje WS-BPEL 2.0, utilizado para el desarrollo de las composiciones, lenguaje que se ha estudiado a través de la especificación facilitada por el tutor y, posteriormente, mediante el estudio de algunas composiciones a modo de ejemplo. En esta fase también se incluye el

estudio de otras tecnologías como WSDL y BPELUnit, además de XML Schema, XPath 2.0, Velocity, SOAP y el lenguaje TestSpec.

Además, se ha realizado el estudio de los distintos operadores de mutación para WS-BPEL 2.0 de los que dispone el grupo de colaboración UCASE, puesto que el desarrollo de las composiciones de este proyecto está destinado a la aplicación específica de dichos operadores.

2.2.3. Elección de herramienta gráfica para el desarrollo de composiciones

En esta fase se llevó a cabo un estudio para elegir qué herramienta gráfica utilizar para el desarrollo de las composiciones WS-BPEL 2.0. Se evaluaron las herramientas de NetBeans y ECLIPSE, puesto que son dos herramientas gráficas muy comunes y utilizadas para el desarrollo de este tipo de software.

Se decidió utilizar la herramienta ofrecida por ECLIPSE debido a diversas razones:

- Además de poder desarrollar con ECLIPSE los ficheros *wsdl* y *bpel*, ofrece la posibilidad de desarrollar los ficheros *bpts* que contienen las pruebas necesarias para la ejecución de las composiciones.
- Ofrece una mayor facilidad a la hora de desarrollar los ficheros *wsdl*.
- Presenta una mayor sencillez a la hora de editar las propiedades de las actividades en su interfaz gráfica.
- Ofrece una mayor comodidad cuando se desean añadir al proceso *parner links*, variables, *correlation sets* y *message exchanges*.
- Su interfaz se muestra algo más intuitiva que la de NetBeans.

2.2.4. Desarrollo de las composiciones WS-BPEL

El desarrollo de composiciones sigue un esquema general. Para cada composición se desarrolla un fichero *bpel* en el que se define la lógica del proceso, un fichero *wsdl* en el que se definen todas las características necesarias para los servicios que se utilizan en la composición. También se desarrolla un fichero *bpts* con los casos de pruebas empleados, así como, otro fichero *bpts* basado en plantillas y los ficheros necesarios para los casos de pruebas basados en plantillas: un fichero *spec* y un fichero *vm* generado a partir de él.

Asimismo, todas las composiciones desarrolladas deben ser integradas en un repositorio de composiciones perteneciente al grupo de investigación UCASE, donde se cuenta con un artículo wiki para cada una de ellas.

Desarrollo de la composición squaresSum_1

En esta etapa se realizó el desarrollo de la primera composición, con la que se pretendía poder aplicar el operador AIS. Por lo que había que dotarla de una estructura que permitiera la aplicación de tal operador.

En concreto, esta composición calcula el cuadrado de cada número entero comprendido entre 0 y "n", así como la suma total de las potencias. Además de los ficheros necesarios que sigue el esquema general comentado anteriormente, esta composición también consta de un fichero *xsd* en el que se define externamente la estructura de los mensajes que son utilizados por los distintos servicios.

Desarrollo de la composición squaresSum_2

El objetivo de esta etapa era el desarrollo de la composición squaresSum_2. Mediante esta composición se pretendía poder aplicar el operador EEU. Así como el operador AIS.

En esta composición, además de calcularse el cuadrado de cada número entero comprendido entre 0 y "n" y la suma total de las potencias, se calcula la varianza de los números enteros entre 0 y "n" y el resultado de x^2/n para cada uno. Al igual que en la composición anterior, también se cuenta con un fichero *xsd* en el que se define externamente la estructura de los mensajes.

Desarrollo de la composición tradeIncome

El desarrollo de esta composición lo podemos subdividir también en distintas fases.

- **Estudio y eliminación del operador ACI**

  El objetivo principal que se intentaba alcanzar con esta composición era la aplicación del operador ACI. Para la aplicación de tal operador había que dotar a la composición de una determinada estructura, que permitiera al operador actuar en la composición de manera que se pudiesen obtener mutantes válidos para su ejecución.

  Pero en vista de la imposibilidad de conseguir que ese operador se aplicara debido a que las restricciones de ActiveBPEL hacían imposible que dicha estructura se pudiera conseguir, a través de esta composición se decidió por la eliminación del operador de mutación ACI de la lista de los operadores de mutación para WS-BPEL 2.0 de los que dispone el grupo de investigación UCASE.

- **Aplicación del operador ECC**

  El siguiente objetivo a alcanzar con este operador era el de poder aplicar el operador ECC. Este operador cambia un operador camino (/ , //) por otro en una expresión. Por lo que para que se puedan generar mutantes que no sean equivalentes es necesario dotar en el fichero *wsdl* a un determinado mensaje, que un servicio utilice, con la estructura necesaria para que al escribir en el fichero *bpel* una expresión que haga referencia a un determinado elemento de ese mensaje, el operador ECC pueda producir esos mutantes que no sean equivalentes.

- **Aplicación de los operadores AWR, AFP, AIS y EEU**

  En esta etapa se pretende dotar a la composición con la estructura necesaria para poder aplicar los operadores AWR, AFP, AIS y EEU.

- **Aplicación de operadores de mutación de condiciones excepcionales y eventos**

  En esta fase se desea dotar a la composición de una estructura para que los operadores de mutación de condiciones excepcionales y eventos (XMF, XMC, XMT, XTF, XER y XEE) puedan ser aplicados. Por lo que es necesario asignar a la composición una lógica en la que sea necesaria la utilización de los manejadores de fallos, manejadores de eventos, manejadores de compensación y manejadores de terminación, así como de las actividades *throw* y *rethrow*.

- **Aplicación de los operadores APA y APM**

  En esta fase se trabaja con los operadores relacionados con los elementos que pueden formar parte de la actividad de recepción de mensajes *pick*. Estos pueden ser los elementos *onMessage* y *onAlarm*, éste último también puede aparecer en un manejador de eventos. Por lo que se necesita dotar a la composición de la lógica necesaria que introduzca estos elementos para que se puedan aplicar los operadores APA y APM, que eliminan los elementos *onMessage* y *onAlarm*, respectivamente.

- **Aplicación de los operadores AJC y ELL**

  El objetivo a alcanzar en esta fase consiste en intentar lograr una sincronización entre actividades mediante los contenedores opcionales del lenguaje WS-BPEL 2.0 *sources* y *targets*, pudiendo contener este último un elemento *joinCondition* con el que se establece una condición, que cuando se cumple, permite la ejecución de la actividad en la que está contenida. De esta manera, se puede aplicar en la composición el operador AJC, que se encarga de eliminar el elemento *joinCondition*.

  También se aplica el operador ELL, que se encarga de cambiar en una expresión un operador lógico (and, or) por otro del mismo tipo.

- **Aplicación del operador EMF**

  El objetivo de esta etapa es el de conseguir aplicar en la composición el operador EMF, que modifica una expresión de fecha límite. Por lo que es necesario que la composición cuente con una estructura que permita la aplicación de tal operador.

En concreto, mediante esta composición se modela la gestión de un supermercado y se define una lógica para el proceso que contiene todas las características comentadas anteriormente. Además de los ficheros que se desarrollan como esquema general para cada composición, consta de otro fichero *wsdl* en el que se definen por separado los *partnerLinkType* y los distintos elementos que se utilizan para la correlación de mensajes.

Desarrollo de la composición squaresSum_3

El objetivo de esta etapa era el desarrollo de la composición squaresSum_3. El principal propósito era el de conseguir que la composición tuviera la estructura necesaria para hacer posible la aplicación de los operadores de cobertura (CFA, CDE, CCO y CDC) y los relacionados con la cobertura (EIU, EIN, EAP y EAN).

En esta composición el desarrollo del fichero *bpel* está orientado a, además de comprender el comportamiento indicado para squaresSum_2, indicar si el número recibido se trata de un número primo o no. Además de los ficheros que se desarrollan como esquema general, se cuenta con un fichero *xsd* donde se define externamente la estructura de los mensajes.

2.2.5. Documentación

Etapa en la que se ha desarrollado la memoria de este PFC. Para ello, se ha empleado el lenguaje LaTeX [13] [14], que facilita el desarrollo de documentos de grandes dimensiones. Además, se han realizado los distintos artículos wiki para el repositorio de las composiciones, donde se describen las composiciones desarrolladas.

2.3. Distribución temporal

En la figura 2.1 podemos observar el Diagrama de Gantt, con el que se presenta una representación gráfica de la distribución temporal de las distintas fases que ha abarcado este PFC.

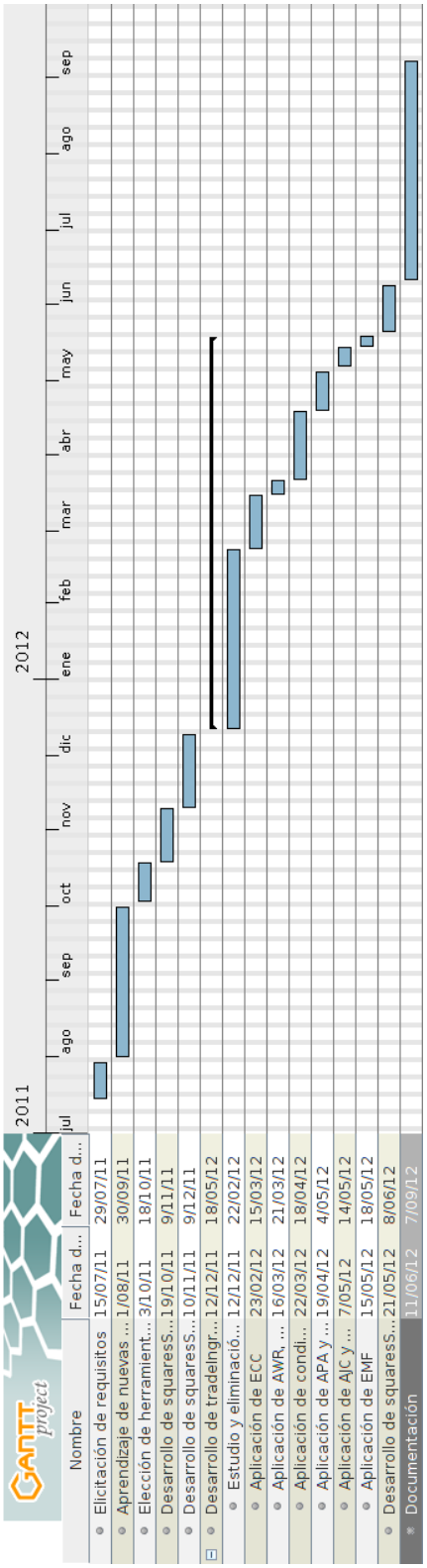

Figura 2.1: Diagrama de Gantt

Capítulo 3

Descripción general del proyecto

En este capítulo del documento se detallan las funciones del proyecto. Además, se especifican las distintas herramientas que han sido necesarias durante la realización del PFC y las restricciones que ha de cumplir.

3.1. Funciones del producto

El desarrollo de las distintas composiciones de servicios web en lenguaje WS-BPEL 2.0 está orientado a una línea de investigación basada en la aplicación de la prueba de mutaciones. De manera que cada composición esté provista de una lógica determinada con la que permita que operadores de mutación específicos puedan aplicarse con cada una de ellas.

En la realización de una composición de servicios web en WS-BPEL. Además del fichero con extensión *bpel* en el que se defina la lógica de negocio, es necesario también, al menos, un fichero *wsdl* en el que se especifiquen los distintos servicios que van a interactuar. Así como, los ficheros con extensión *bpts* en los que se definan los casos de prueba contra los que ejecutar la composición y los mutantes generados a partir de la composición, comprobando a través de los resultados obtenidos si los casos de pruebas utilizados son suficientes o son necesarios la adición de otros con los que detectar a todos los mutantes no equivalentes. Se realizan también casos de pruebas mediante la utilización de plantillas Velocity, con las que se consigue automatizar la generación de los casos de prueba. Los datos que son utilizados en estos casos de pruebas basados en plantillas pueden encontrarse en el mismo fichero o generarse a través de un fichero con extensión *spec* escrito en lenguaje TestPec, que contiene la definición y declaración de los tipos de datos necesarios y, mediante el que se genera un fichero con extensión *vm* que contiene los datos que se intercambiarán en los mensajes de cada caso de prueba. También es necesario cuando se desarrolla una composición utilizar otros lenguajes como XPath, con el que podemos realizar consultas mediante expresiones situadas en el propio código WS-BPEL, XML Schema, con el que se define la estructura de los mensajes con los que interactúan los distintos servicios, y SOAP, empleado conjuntamente con WSDL.

Para la realización de composiciones web en WS-BPEL se utilizan una serie de herramientas que facilitan su desarrollo, como ECLIPSE. Así como herramientas que ayuden en la ejecución y comparación de composiciones y mutantes para poder llevar a cabo la prueba de mutaciones, como MuBPEL, que permitan la visualización de los resultados obtenidos en la ejecución de las composiciones, como TkDiff, y que puedan identificar las diferencias existentes entre los distintos mutantes y composiciones, como XMLEye.

3.2. Herramientas

A continuación, se detallan las distintas herramientas que han sido utilizadas durante la realización de este proyecto.

3.2.1. MuBPEL

MuBPEL [3] es la herramienta utilizada, creada por el grupo de investigación UCASE, para la ejecución de las composiciones WS-BPEL. Así como para realizar la comparación entre la composición y los mutantes obtenidos a partir de ella, generar los mutantes o analizar cuantos mutantes se pueden obtener de una composición, entre otras cosas.

Todas las funciones de MuBPEL son las siguientes. Las cuales podemos obtener al escribir en la terminal:

```
mubpel -h
```

```
Available subcommands:
 * analyze bpel
 * apply bpel operator operandIndex attribute
 * applyall bpel
 * compare bpts bpel xml (bpel1...|-)
 * comparefull bpts bpel xml (bpel1...|-)
 * compareout xml xml1...
 * normalize bpel
 * run bpts (bpel1...|-)
```

3.2.2. TkDiff

TkDiff [15] es una herramienta gráfica con la que podemos observar las diferencias existentes entre dos ficheros. Esta herramienta muestra, a pantalla compartida, los dos ficheros marcando línea a línea las diferencias encontradas entre ambos.

De este modo, podemos utilizar TkDiff para comparar un fichero original *bpel* con un mutante generado por un determinado operador de mutación y, así, observar qué parte del código se ha modificado y estudiar el resultado obtenido al ejecutarlo contra el conjunto de casos de prueba utilizado.

Para realizar una comparación entre dos ficheros podemos escribir directamente la siguiente orden en una terminal.

```
tkdiff fichero1.bpel fichero2.bpel
```

En la figura 3.1 podemos observar la interfaz de TkDiff.

3.2.3. XMLEye

XMLEye [16] es un visor genérico de documentos basados en XML. No está ligado a ningún formato en particular, sino que puede abrir casi cualquier tipo de documento, que debe ser transformado previamente a XML para poder visualizarse.

Esta herramienta es muy útil para visualizar las salidas que producen las composiciones al ejecutarlas contra los casos de prueba y comprobar que todos se han ejecutado correctamente.

Para visualizar un documento con XMLEye podemos escribir directamente la siguiente orden en una terminal.

```
xmleye salida.xml
```

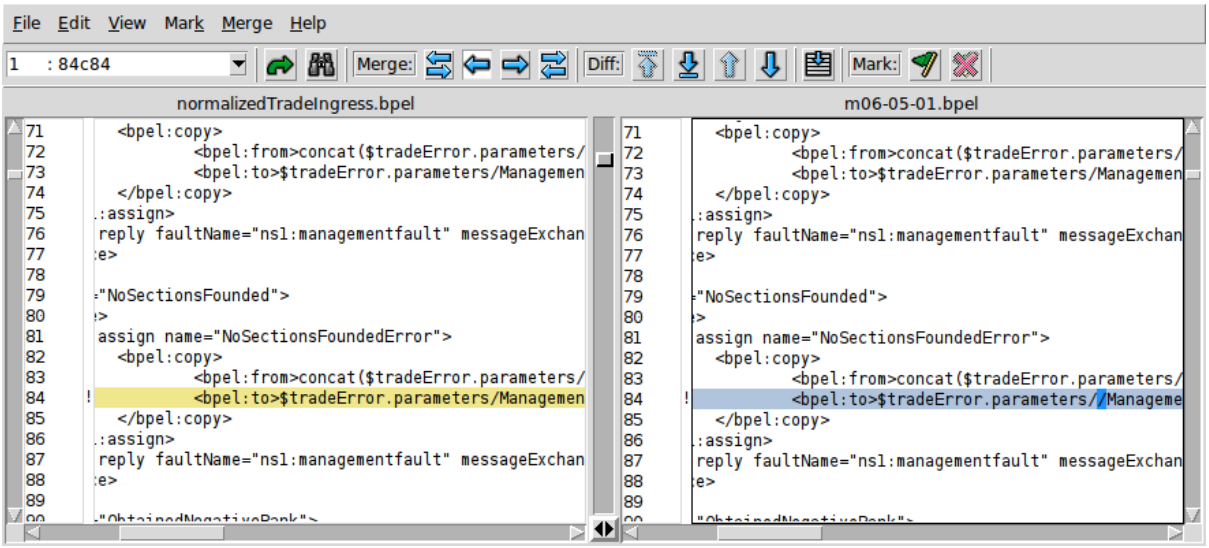

Figura 3.1: Herramienta TkDiff

En la figura 3.2 podemos observar la interfaz de XMLEye.

3.2.4. ActiveBPEL

ActiveBPEL [17] es un motor con el que se pueden ejecutar los procesos WS-BPEL y que BPELUnit puede utilizar en el momento de sustituir por servicios emulados los servicios externos. Esta sustitución se puede desear hacer por diversos motivos:

- Los servicios reales no están disponibles, o no queremos interactuar con ellos.
- Los servicios reales bloquean recursos.
- Se desea establecer de antemano cuáles van a ser las respuestas o peticiones de los servicios externos, de manera que sea posible controlar el entorno de prueba en todo momento.

3.2.5. ECLIPSE

ECLIPSE [18] es la herramienta utilizada para realización de las composiciones. Se trata de un entorno de desarrollo integrado multiplataforma para el que existe un gran número de módulos con los que se permite el desarrollo bajo, prácticamente, cualquier lenguaje de programación.

ECLIPSE ofrece una interfaz gráfica que facilita el desarrollo tanto de los ficheros *bpel*, como de los *wsdl* y *bpts*. Aunque también dispone de un editor de texto con resaltado de sintaxis que permite modificar directamente de una manera sencilla el código fuente.

En la figura 3.3 podemos observar la interfaz de ECLIPSE.

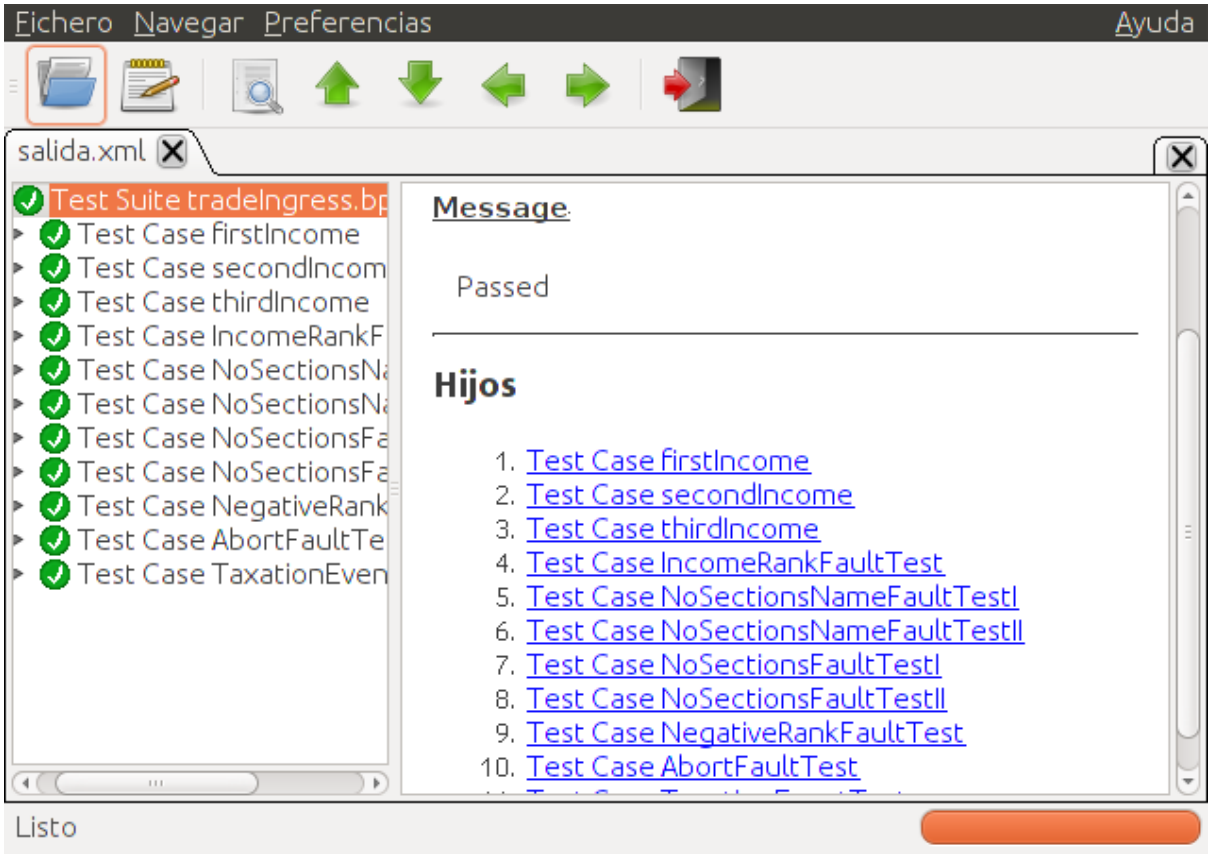

Figura 3.2: Herramienta XMLEye

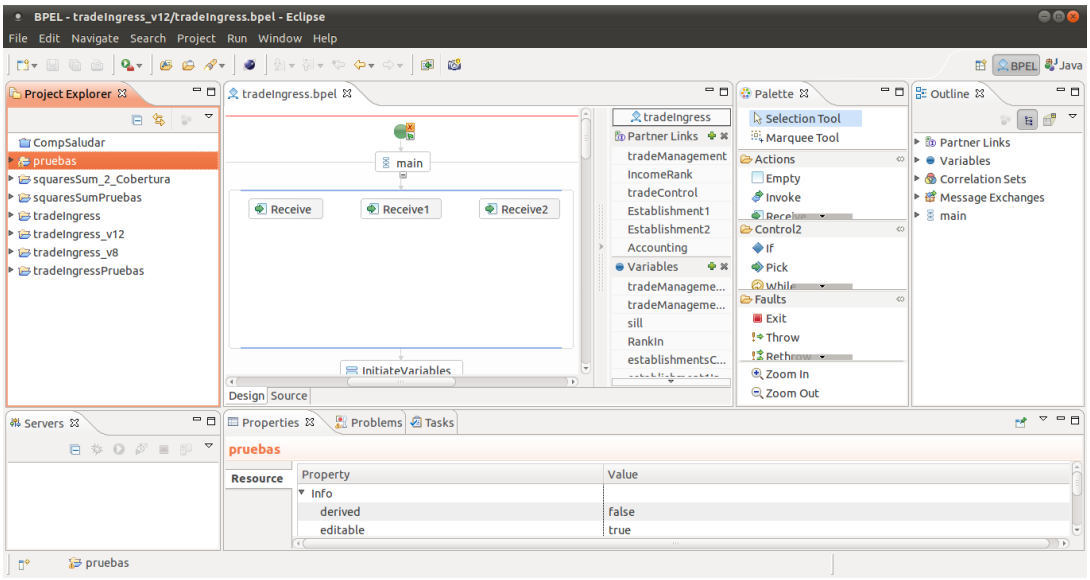

Figura 3.3: Herramienta ECLIPSE

3.3. Restricciones generales

3.3.1. Restricciones del repositorio

Las composiciones WS-BPEL desarrolladas pasarán a formar parte de un repositorio que utiliza el grupo de investigación UCASE. Este repositorio tiene una serie de restricciones que tienen que cumplir todas las composiciones que formen parte de él.

Las composiciones deben estar constituidas por lo siguiente:

- Un fichero con extensión *bpel* que contenga la lógica de la composición.
- Uno o más ficheros con extensión *bpts* con los casos de pruebas disponibles para la composición. Al menos uno de los ficheros *bpts* debe utilizar plantillas Velocity para definir los casos de prueba a ejecutar de acuerdo a un fichero externo. La composición ha de pasar correctamente todos los casos de prueba de cada fichero *bpts* sin producir ningún error.
- Al menos un fichero de datos con extensión *vm* o *csv* para que sean utilizados por los ficheros *bpts* con los casos de prueba basados en plantillas. Estos ficheros deben ser generados a partir de un fichero con extensión *spec* que defina el formato de los datos que deben contener.
- Todos los ficheros con extensión *wsdl* y *xsd* importados por el fichero *bpel*.

Por el contrario, deben estar exentas de lo detallado a continuación:

- Ficheros temporales como, por ejemplo, los * producidos por algunos editores.
- Archivos de proyectos Eclipse/NetBeans.
- Archivos comprimidos, como *.bpr o *.zip.

Además, las composiciones deben ser compatibles con el Perfil Básico de Interoperabilidad de Servicios Web [19], considerando la estructura de los mensajes. Una forma rápida de comprobarlo es ejecutar "service-analyzer *.wsdl" y observar si se produce un catálogo de mensajes correctamente. También se

puede comprobar manualmente, asegurándose de que las operaciones en las que se usa el estilo *rpc* en el elemento *soap:binding* usen el atributo *type* para las partes de sus mensajes, y las operaciones en las que se usa el estilo *document* usan el atributo *element* para las partes de sus mensajes.

Los casos de prueba incluidos en los archivos *bpts* deben tener exhaustivas condiciones de prueba para asegurar que la composición trabaja como se espera.

En el repositorio utilizado por el grupo UCASE, para cada composición será también será necesario un artículo en la Wiki, en el que se describan las distintas características de la composición, como su comportamiento, casos de prueba, autor, etc.

Podemos visitar el repositorio en la siguiente dirección: `https://neptuno.uca.es/redmine/projects/wsbpel-comp-repo`

3.3.2. Control de versiones

Durante el desarrollo de este proyecto, se ha utilizado un sistema de control de versiones, que proporciona ventajas como la protección de los datos que se tienen del proyecto, así como, la posibilidad de deshacer cambios erróneos en el proyecto, al disponer de todas las versiones previas del mismo.

El sistema de control de versiones utilizado ha sido Subversion [20]. Se trata de un sistema centralizado utilizado frecuentemente por desarrolladores.

Las órdenes que pueden ser de mayor interés para trabajar con Subversion son las siguientes:

- Podemos obtener una copia de un repositorio existente mediante:

```
svn co URLrepositorio
```

- Podemos actualizar la copia del repositorio escribiendo:

```
svn update
```

- Podemos añadir un fichero al sistema de control de versiones mediante:

```
svn add fichero
```

- Podemos guardar los cambios que hayamos realizados mediante:

```
svn ci -m "Cambios realizados"
```

  Con la opción -m se crea un mensaje con el que se puede indicar los cambios que se han realizado.

En el manual de Subversion podemos encontrar más información sobre las instrucciones que se pueden utilizar para trabajar con este sistema de control de versiones.

Capítulo 4

Desarrollo del proyecto

En este capítulo se detalla la fase de desarrollo del PFC. Por tanto, se realiza el análisis y diseño del proyecto, desarrollándose para ello el diagrama y la especificación de casos de usos, así como el modelo conceptual de datos, el modelo de comportamiento del sistema y el diagrama de secuencia correspondientes. Tanto en la parte de análisis como en la de diseño nos centramos en la composición tradeIncome, puesto que es la más compleja de las composiciones desarrolladas.

4.1. Herramienta de modelado *DIA*

DIA es una herramienta bastante sencilla de usar con la que podemos crear diferentes tipos de diagramas, como diagramas UML, diagramas de flujo, diagramas de entidad-relación, etc.

Presenta una interfaz bastante amigable e intuitiva, consta de un panel de objetos que puede cambiar según el tipo de diagrama que elijamos. Los objetos de este panel de control, sencillamente, se arrastran al documento en el que estemos trabajando para formar el esquema que deseemos. Además, se puede personalizar dicho panel de objetos, eligiendo entre los disponibles cuales se desea que se visualicen o, incluso, añadiendo nuevos objetos a partir de un archivo con extensión *svg*.

Por otro lado, ofrece una serie de herramientas que facilitan el desarrollo de cualquier diagrama, como, por ejemplo, reglas y una cuadrícula a la que se pueden ajustar los distintos objetos.

Mediante esta herramienta, los diagramas que han sido creados pueden ser exportados a diferentes formatos, como svg, png, eps, e incluso a código LaTeX.

También se pueden exportar los diagramas directamente mediante la generación de código C++, Java, Python y Pascal.

En la figura 4.1 podemos observar la interfaz de DIA.

4.2. Análisis

En la parte de análisis se trata de conseguir la especificación detallada del sistema, concretando los requisitos que se necesitan para satisfacer unas determinadas necesidades. Para ello se desarrollará una serie de procesos con los que se define la estructura conceptual del sistema. Tales procesos son el modelo de casos de uso y el diagrama conceptual.

4.2.1. Modelo de casos de uso

Los casos de uso son una técnica de especificación de requisitos apropiada tanto en desarrollos estructurados como en desarrollos orientados a objetos, siendo especialmente conveniente en este último caso, ya que sirve como referencia a lo largo de todo el ciclo de vida.

Se realiza el modelo de casos de uso identificando:

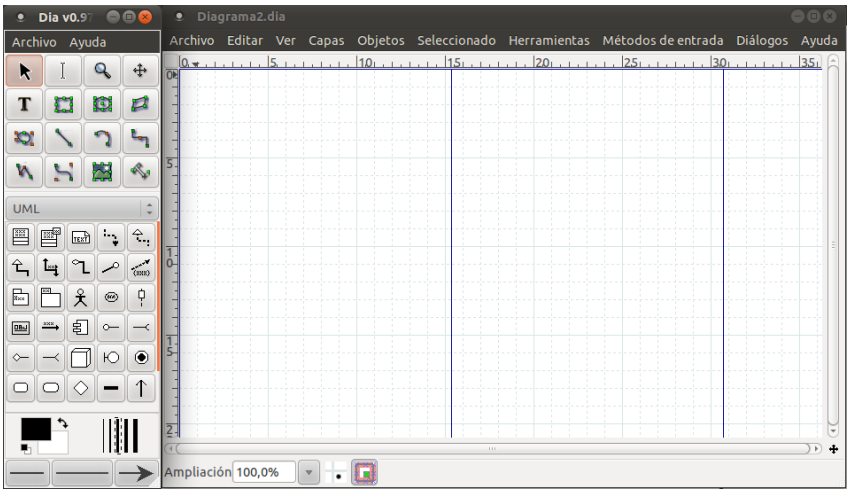

Figura 4.1: Herramienta DIA

Figura 4.2: Diagrama de casos de uso

- Actores: Un actor representa un conjunto coherente de roles que interpretan los usuarios de los casos de uso al interaccionar con el sistema.
- Casos de uso: Un caso de uso describe un conjunto de secuencias de interacciones entre actores y el sistema. Con un caso de uso se describe un comportamiento deseado del sistema.
- Descripción de cada caso de uso: Dónde se indican el escenario principal y los posibles escenarios alternativos.

Diagramas de casos de uso

En la figura 4.2 podemos observar el diagrama de casos de uso. Como vemos, tenemos el caso de uso “Gestionar supermercado” y un actor, el usuario, que es el encargado de solicitar la gestión del supermercado.

Especificación de casos de uso

Se realiza la especificación de cada caso de uso, mediante la definición para cada caso de uso de los siguientes apartados:

- Actor.
- Descripción del escenario.
- Precondiciones y postcondiciones.

- Escenario principal.
- Escenarios secundarios.

Caso de uso: Gestionar supermercado

Actor: El cliente, que es la entidad del supermercado interesada en conocer los detalles de la gestión del mismo.
Descripción: Se realiza la gestión del supermercado para una determinada fecha.
Precondiciones: El cliente introduce una fecha correcta.
Postcondiciones: El sistema proporciona los datos relacionados con la gestión del supermercado.
Escenario principal:

1. El cliente desea realizar la gestión del supermercado para una determinada fecha.
2. El sistema espera por un mensaje del cliente con la fecha de la que se desea la gestión y un mensaje de cada supermercado con los datos relativos a ingresos, costes y stock disponible hasta esa fecha.
3. El sistema recibe un mensaje que contiene las ofertas disponibles, como estrategia de marketing, y calcula el número de elementos al que afectarán las ofertas.
4. El sistema realiza el cálculo del balance con los datos de los dos establecimientos.
5. El sistema recibe un mensaje que indica la cantidad a partir de la cual se calcula si se obtienen beneficios.
6. El sistema determina si la gestión ha sido rentable y envía un mensaje confirmando que los datos de los supermercados se han utilizado correctamente para realizar el balance a la entidad encargada del control de los establecimientos.
7. El sistema puede indicar al cliente que se debe realizar el informe anual si se ha alcanzado la fecha establecida para ello.
8. El sistema aplica los posibles costes adicionales que pudiera haber a las ganancias totales.
9. El sistema envía un mensaje con todos los datos calculados.

Escenario alternativo:

- 3.a: El sistema recibe un mensaje que contiene la publicidad a realizar, como estrategia de marketing, y lo almacena en la variable de salida.
- 3.b: El sistema no recibe ningún mensaje relacionado con la estrategia de marketing.
- 4.a: El sistema detecta que se ha producido un error al intentar realizar el balance de uno de los establecimientos debido a que, en el mensaje recibido, el establecimientos no tiene secciones.
    1. El sistema lanza una excepción indicando el error ocurrido.
    2. El sistema finaliza el cálculo del balance.
    3. El sistema capta dicha excepción y envía un mensaje indicando el fallo.
- 4.b: El sistema detecta que se ha producido un error al intentar realizar el balance de uno de los establecimientos debido a que, en el mensaje recibido, algunas secciones carecen del atributo *name*.

1. El sistema lanza una excepción indicando el error ocurrido.
2. El sistema finaliza el cálculo del balance.
3. El sistema capta dicha excepción y envía un mensaje indicando el fallo.

- 5.a: El sistema recibe un fallo en lugar del mensaje con la cantidad a partir de la cual se obtienen beneficios.

  1. El sistema capta dicha excepción y envía un mensaje indicando el fallo.

- 6.a: El sistema detecta que la cantidad recibida es negativa y lanza una excepción indicando el error ocurrido.

  1. El sistema capta dicha excepción.
  2. El sistema envía un mensaje a la entidad encargada del control de los establecimientos indicando que se ha producido un error.
  3. El sistema envía un mensaje indicando el fallo.

- 2-7.a: El cliente decide abortar la gestión.

- 2-7.b: El sistema recibe un mensaje indicando que se deben incluir costes adicionales en la gestión del supermercado.

  1. El sistema prosigue por el punto en el que esté.

4.2.2. Modelo conceptual de datos

Con el modelo conceptual de datos se pueden modelar los requisitos de datos de un sistema. En él se describen las estructuras de datos de un sistema y las relaciones que pueden existir entre ellos.

Un modelo conceptual de datos normalmente contiene:

- Clases de objetos.
- Asociaciones entre clases de objetos.
- Atributos de las clases de objetos.
- Restricciones de integridad.

En la figura 4.3 podemos encontrar el modelo conceptual de datos de nuestro sistema. El gestor del supermercado desea realizar un gestión del mismo a una determinada fecha, a través de esta gestión se obtiene un balance en el que se conocen los detalles de tal gestión. Dicha gestión se puede llevar a cabo gracias al empleo de los datos de la recaudación proporcionada por los distintos establecimientos, además de, la consideración de una cantidad límite a partir de la cual se puede decidir si el balance ha resultado ser beneficioso o no para el supermercado. El balance también se ciñe a la aplicación de una posible tributación adicional, así como, a la aplicación de una posible estrategia de marketing a seguir, como puede ser la realización de determinadas ofertas o el desarrollo de una serie de anuncios en un determinado medio de comunicación.

También, se cuenta con una entidad encargada de controlar si la información que ha sido proporcionada por los establecimientos se ha utilizado correctamente a la hora de realizar el balance o, por el contrario, ha ocurrido un error determinado.

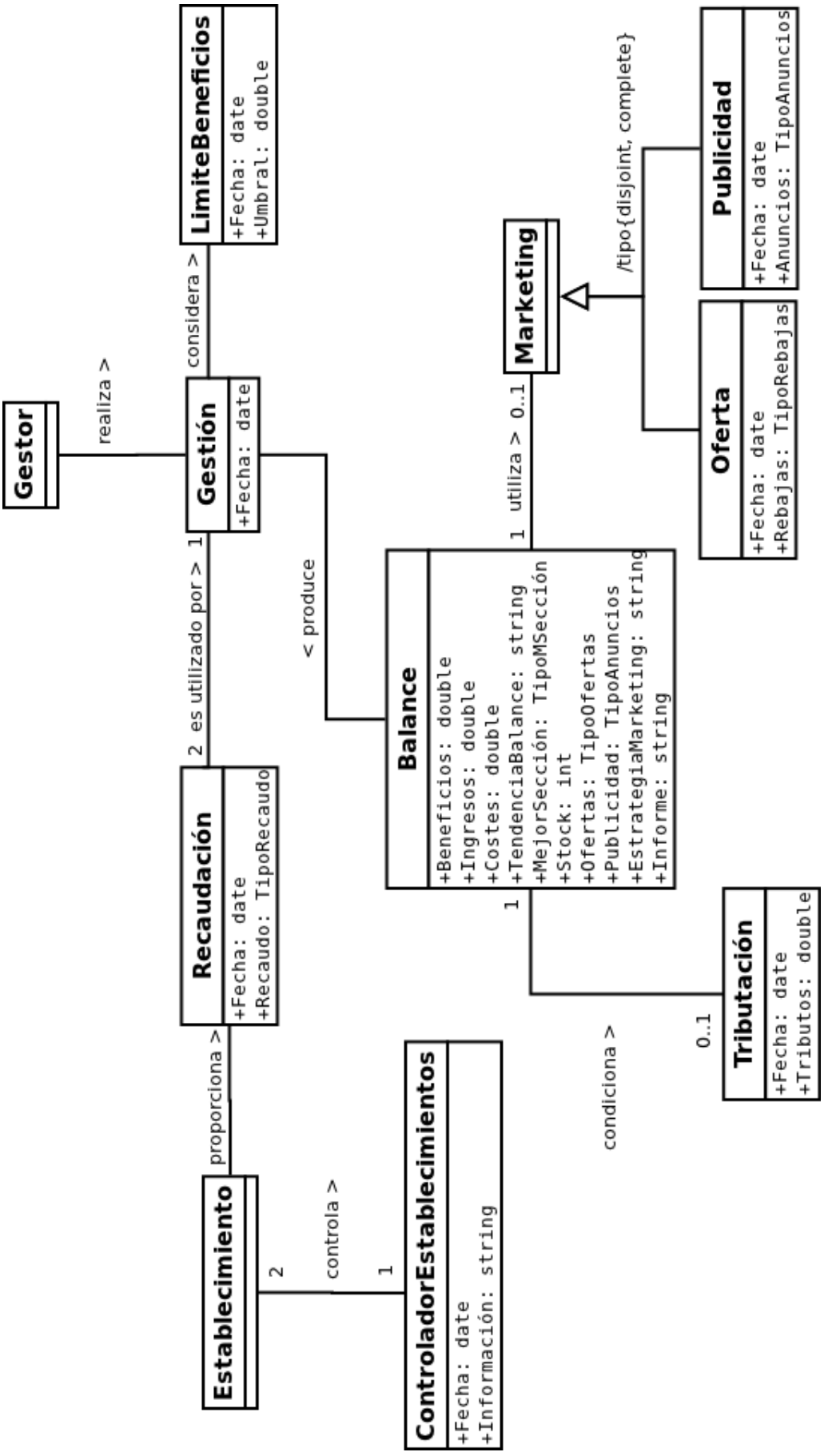

Figura 4.3: Modelo Conceptual de Datos

4.2.3. Modelo de comportamiento del sistema

El modelo de comportamiento del sistema nos permite obtener la especificación del comportamiento. Se utilizan dos técnicas para llevar a cabo dicho modelo: los diagramas de secuencia y los contratos de las operaciones. Estas técnicas se aplican a los casos de usos de los que se compone el sistema.

Los diagramas de secuencia muestran la secuencia de eventos entre los actores y el sistema y nos permiten identificar las operaciones del sistema. Mientras que con los contratos de las operaciones del sistema podemos describir el efecto de las mismas.

Caso de uso: Gestionar supermercado

Diagrama de comportamiento.

En la figura 4.4 podemos encontrar el diagrama de comportamiento para el caso de uso *Gestionar supermercado*.

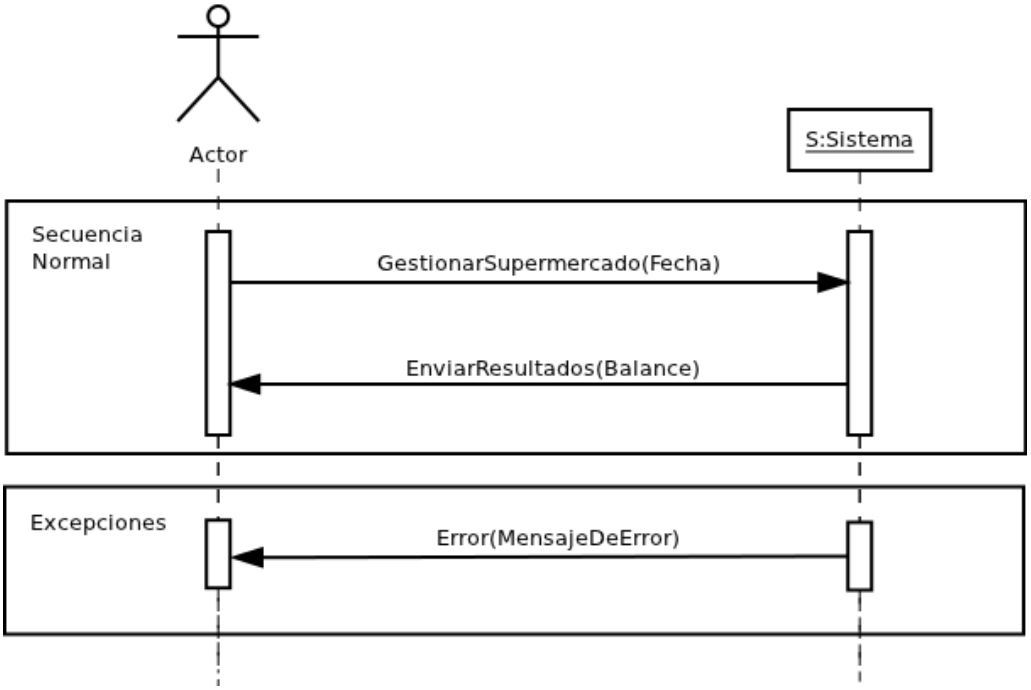

Figura 4.4: Diagrama de comportamiento

Contrato de las operaciones.

Operación: GestionarSupermercado(Fecha)

Responsabilidades: Realiza la gestión del supermercado a la fecha proporcionada.

Precondiciones: El cliente introduce una fecha correcta.

Postcondiciones: Como resultado de la gestión del supermercado se crea una instancia de un balance con los datos de tal gestión que se envía al cliente.

4.3. Diseño

En la parte de diseño, se muestra el diagrama de secuencia de la composición tradeIncome. Además, se detalla como interactúan entre sí los distintos ficheros necesarios para el correcto funcionamiento

de una composición WS-BPEL, así como, la forma en la que la herramienta MuBPEL nos es útil a la hora de trabajar con las distintas composiciones, pudiendo ejecutarlas, realizar un análisis, generar los posibles mutantes y comparar sus ejecuciones.

4.3.1. Diagrama de secuencia para la composición tradeIncome

El diagrama de secuencia permite describir cómo los objetos colaboran entre sí para realizar una determinada actividad, destacando la ordenación temporal de los mensajes. Este tipo de diagramas incluyen los objetos y clases que intervienen en el desarrollo de una actividad, definiendo los mensajes que se intercambian entre ellos y en qué orden, para poder realizar con éxito la actividad deseada.

En la figura 4.5 podemos encontrar el diagrama de secuencia de nuestro sistema.

4.3.2. Descripción de la arquitectura

Las composiciones en WS-BPEL 2.0, además del fichero *bpel* en el que se define el comportamiento del proceso de negocio, necesitan del desarrollo de otros ficheros para el correcto funcionamiento de éstas.

Por un lado tenemos los ficheros *wsdl*. Un proceso de negocio necesita interactuar con unos determinados socios de negocio, esta interacción ocurre gracias a las interfaces de los servicios web, que son descritos en estos ficheros. De modo, que el proceso de negocio hace uso de estos servicios descritos en los ficheros *wsdl*, definiendo en su comportamiento la coordinación de las distintas interacciones entre el proceso y los socios de negocio.

Por otro lado, tenemos los ficheros *bpts*, con los que podemos definir un conjunto de casos de prueba contra los que ejecutar la composición. Estos casos de prueba están formados por las actividades con las que va a interactuar el proceso y los distintos socios de negocio, conteniendo los mensajes que va a enviar o recibir cada socio, con todos los datos necesarios y una serie de condiciones a aplicar sobre los datos recibidos, para controlar que el proceso se ha ejecutado correctamente.

Además, contamos con ficheros *bpts* basados en plantillas Velocity con los que se puede automatizar la generación de los casos de prueba. En este sentido, contamos con un fichero *spec*, en el que se describe el formato de los datos que van a generarse para poder ser utilizados por los casos de pruebas, y con un fichero *vm*, que contendrá los datos generados aleatoriamente a partir del fichero *spec*.

La herramienta MuBPEL es la encargada de ejecutar estas composiciones para obtener la salida correspondiente.

En la figura 4.6 podemos observar la interacción entre los distintos ficheros implicados en una composición y la obtención de una salida a través de su ejecución con MuBPEL.

Con MuBPEL, además de ejecutar una composición WS-BPEL contra un conjunto de casos de pruebas para obtener su salida, podemos realizar otras acciones como: el análisis del fichero *bpel* para obtener para cada operador de mutación en cuántas localizaciones del código se puede aplicar y con cuántos atributos, la generación de los mutantes aplicables a una determinada composición, así como, la comparación entre las salidas producidas por los mutantes generados y la de la composición original para un determinado conjunto de casos de prueba.

En la figura 4.7 se muestra la arquitectura de MuBPEL. Como podemos observar, MuBPEL consta de un analizador, un generador de mutantes y un sistema de ejecución.

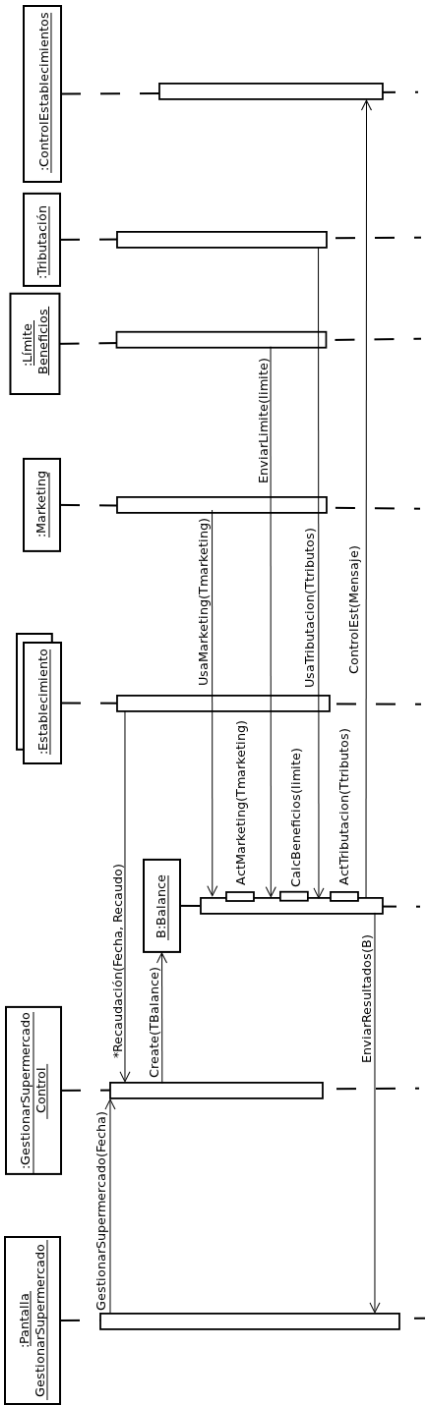

Figura 4.5: Diagrama de secuencia

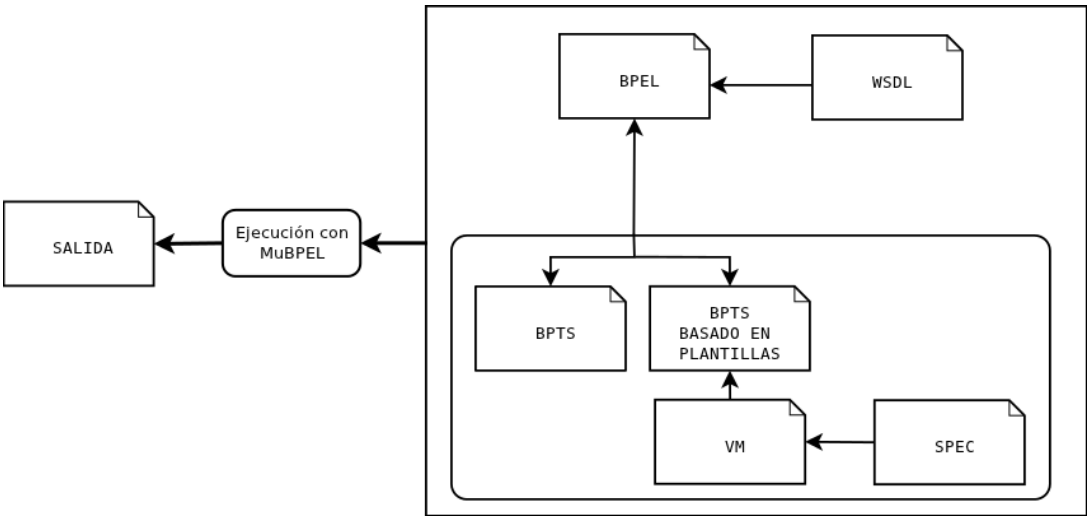

Figura 4.6: Ejecución de una composición

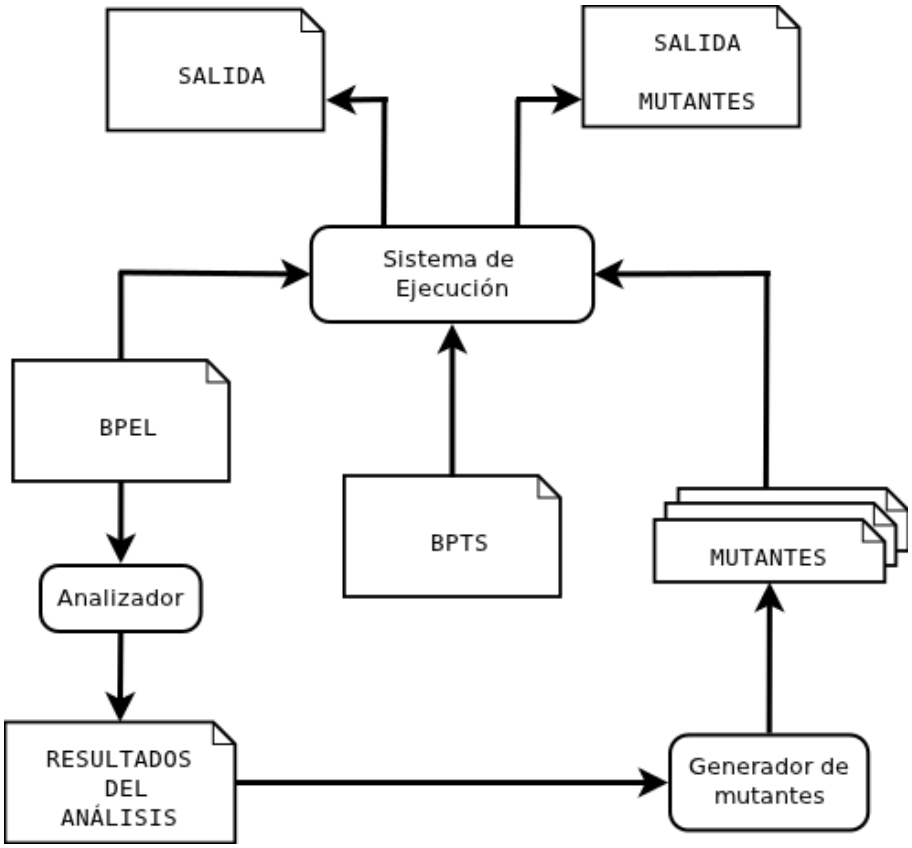

Figura 4.7: Arquitectura de MuBPEL

Capítulo 5

Descripción de las composiciones realizadas

En este capítulo del documento se recogen las distintas composiciones que se han realizado. Se detalla, para cada una de ellas, la función que realiza, su comportamiento, los casos de prueba contra los que se ejecutan, los objetivos para los cuales fue desarrollada y el total de operadores que se aplican en la composición.

5.1. Composición squaresSum_1

En la composición squaresSum_1 se recibe un mensaje con un número entero "n" y se devuelve un mensaje con la potencia al cuadrado de cada número entero comprendido entre 0 y "n" y la suma total de dichas potencias. Para ello, se utiliza un bucle foreach paralelo para calcular la potencia al cuadrado de cada número y otro bucle foreach, también paralelo, para obtener la suma total de las potencias mediante el recorrido de cada elemento calculado.

El mensaje de respuesta utilizado consta de un elemento llamado *sum* con el que se representa la suma de potencias y un elemento *elements*, que consta de otros elementos *element* con los que se representan la potencia al cuadrado de cada número.

El comportamiento de esta composición es el siguiente:

1. Recibe un mensaje con un número "n".
2. Mediante un bucle foreach se calcula la potencia al cuadrado de cada número comprendido entre 0 y "n".
3. Mediante un bucle foreach se recorren los elementos calculados anteriormente y se obtiene la suma total de potencias.
4. Se envía un mensaje de respuesta con los elementos calculados.

En esta composición sólo hay un partner, que actúa como cliente. Los casos de prueba están incluidos en un fichero *bpts*. Son los siguientes:

- ZeroIterations: El cliente manda un mensaje que contiene un número n = 0. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 0, que el número de elementos es 1, y que la suma del contenido de los elementos es 0.
- OneIteration: El cliente manda un mensaje que contiene un número n = 1. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 1, que el número de elementos es 2, y que la suma del contenido de los elementos es 1.

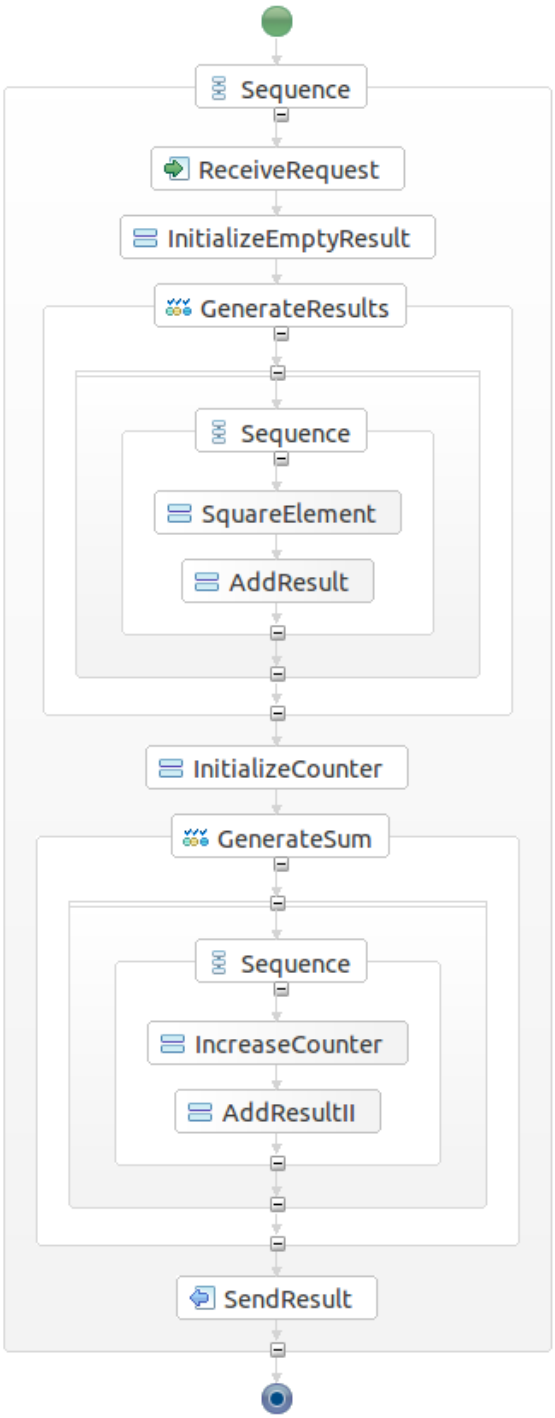

Figura 5.1: Comportamiento squaresSum_1

- SeveralIterations: El cliente manda un mensaje que contiene un número n = 100. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 338350, que el número de elementos es 101, y que la suma del contenido de los elementos es 338350.

El objetivo principal que se intenta conseguir con esta composición es la aplicación del operador AIS, aunque se aplican varios operadores más además del operador AIS. Los operadores que se aplican en esta composición se pueden observar en la tabla 5.1. Dicha tabla consta de tres columnas, la primera muestra el índice correspondiente a cada operador, la segunda representa el nombre del operador y la tercera indica el número de puntos concretos de la composición donde se puede aplicar cada operador.

En total, con esta composición se generan 106 mutantes. De los cuales, con el conjunto de casos de pruebas empleado, 94 mueren, 2 son no válidos y 10 quedan vivos, siendo mutantes equivalentes todos los que quedan vivos. Por lo que el conjunto de casos de prueba empleado mata a todos los mutantes no equivalentes.

5.2. Composición squaresSum_2

Con la composición squaresSum_2, además de obtener la suma total de potencias y la potencia al cuadrado de cada número entero comprendido entre 0 y "n", se obtiene la varianza de los elementos entre 0 y "n" y el resultado de x^2/n para cada uno de ellos.

Para llevar a cabo dicho comportamiento, se utiliza un bucle foreach paralelo para calcular la potencia al cuadrado de cada número, la suma total de potencias y el resultado de x^2/n para cada número. También se utiliza otro bucle foreach paralelo para recorrer y sumar cada elemento calculado para x^2/n en el bucle anterior y poder obtener la varianza mediante la resta de la media al cuadrado.

El mensaje de respuesta utilizado consta de un elemento llamado *sum* que representa la suma de potencias, un elemento *variance*, que representa la varianza, un elemento *elements*, que consta de otros elementos *element* y representan la potencia al cuadrado de cada número, y otro elemento *varianceElements*, que contiene otros elementos *element* para representar el resultado de x^2/n de cada número.

El comportamiento de esta composición es el siguiente:

1. Recibe un mensaje con un número entero "n".

2. Mediante un bucle foreach se calcula la potencia al cuadrado de cada número comprendido entre 0 y "n", la suma total de potencias y cada uno de los elementos de *varianceElements*.

3. Mediante otro bucle foreach se recorren y suman todos los elementos de *varianceElements* calculados anteriormente.

4. Se resta, al cálculo anterior, la media al cuadrado para obtener la varianza.

5. Se envía un mensaje de respuesta con los elementos calculados.

En esta composición sólo hay un partner, que actúa como cliente. Los casos de prueba están incluidos en un fichero bpts. Son los siguientes:

- ZeroIterations: El cliente manda un mensaje que contiene un número n = 0. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 0, que la varianza es 0, que el número de elementos contenidos en *elements* es 1, que el número de elementos contenidos en *varianceElements* es 1, que la suma los elementos contenidos en "elements" es 0 y que la suma de los elementos contenidos en *varianceElements* es 0.

Índice	Operador	Localizaciones
1	ISV	10
2	EAA	4
3	EEU	0
4	ERR	0
5	ELL	0
6	ECC	6
7	ECN	4
8	EMD	0
9	EMF	0
10	AFP	0
11	ASF	0
12	AIS	2
13	AIE	0
14	AWR	0
15	AJC	0
16	ASI	12
17	APM	0
18	APA	0
19	XMF	0
20	XMC	0
21	XMT	0
22	XTF	0
23	XER	0
24	XEE	0
25	AEL	13
26	EIU	6
27	EIN	0
28	EAP	6
29	EAN	6
30	CFA	13
31	CDE	0
32	CCO	0
33	CDC	0

Tabla 5.1: Operadores aplicados en squaresSum_1

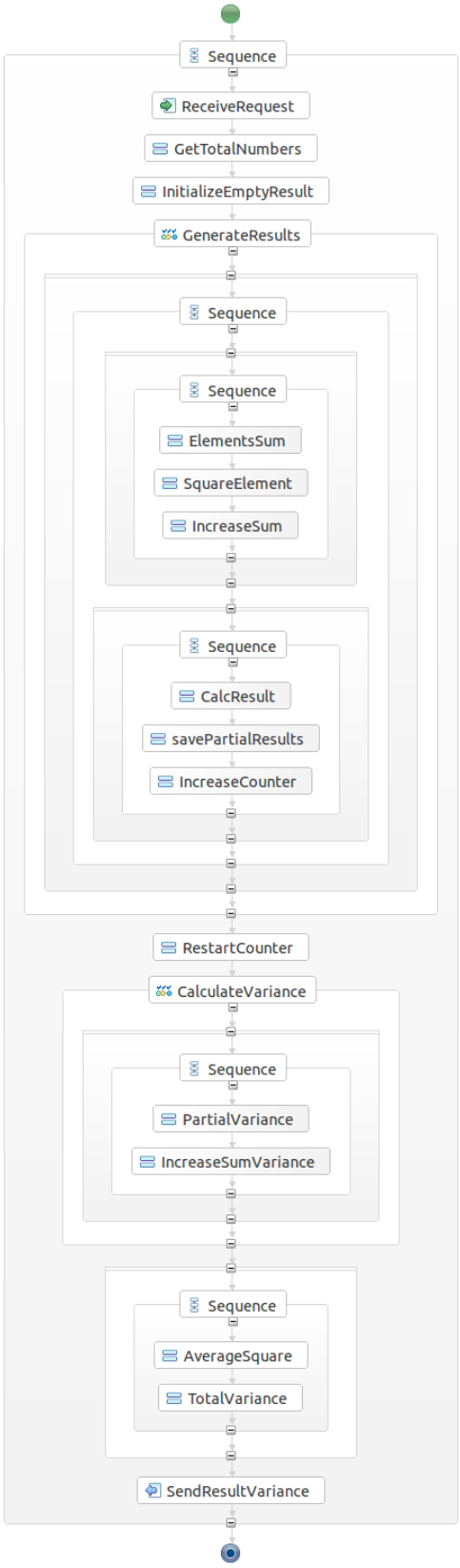

Figura 5.2: Comportamiento squaresSum_2

- OneIteration: El cliente manda un mensaje que contiene un número n = 1. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 1, que la varianza es 0.25, que el número de elementos contenidos en *elements* es 2, que el número de elementos contenidos en *varianceElements* es 2, que la suma los elementos contenidos en *elements* es 1 y que la suma de los elementos contenidos en *varianceElements* es 0.5.

- SeveralIterations: El cliente manda un mensaje que contiene un número n = 100. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 338350, que la varianza es 850, que el número de elementos contenidos en *elements* es 101, que el número de elementos contenidos en *varianceElements* es 101, que la suma los elementos contenidos en *elements* es 338350 y que la suma de los elementos contenidos en *varianceElements* es 3350.000000000001.

El objetivo principal que se intenta conseguir con esta composición es la aplicación de los operadores AIS y EEU, aunque además de estos se aplican algunos operadores más. Los operadores que se aplican en esta composición se pueden observar en la tabla 5.2.

En total, con esta composición se generan 273 mutantes. De los cuales, con el conjunto de casos de pruebas empleado, 242 mueren, 2 son no válidos y 29 quedan vivos, siendo mutantes equivalentes todos los que quedan vivos. Por lo que el conjunto de casos de prueba empleado mata a todos los mutantes no equivalentes.

5.3. Composición squaresSum_3

Esta composición es una extensión de squaresSum_2 mediante la que se intentan aplicar los distintos operadores de cobertura y los relacionados con la misma: EIU, EIN, EAP, EAN, CFA, CDE, CCO y CDC.

En esta composición además del comportamiento modelado en squaresSum_2 se añade el de comprobar si el número que se obtiene mediante el mensaje de entrada es primo o no.

El comportamiento de esta composición es el siguiente:

1. Recibe un mensaje con un número entero “n”.
2. Mediante un bucle foreach se calcula la potencia al cuadrado de cada número comprendido entre 0 y “n”, la suma total de potencias y cada uno de los elementos de *varianceElements*.
3. Mediante otro bucle foreach se recorren y suman todos los elementos de *varianceElements* calculados anteriormente.
4. Se resta, al cálculo anterior, la media al cuadrado para obtener la varianza.
5. Se calcula si el número “n” es primo o no.
6. Se envía un mensaje de respuesta con los elementos calculados.

En esta composición sólo hay un partner, que actúa como cliente. Los casos de prueba están incluidos en un fichero bpts. Son los siguientes:

- ZeroIterations: El cliente manda un mensaje que contiene un número n = 0. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 0, que la varianza es 0, que el número de elementos contenidos en *elements* es 1, que el número de elementos contenidos en *varianceElements* es 1, que la suma los elementos contenidos en *elements* es 0 y que la suma de los elementos contenidos en *varianceElements* es 0. Además, se comprueba que el valor del elemento *prime* sea “false” ya que el número recibido en el mensaje no es primo.

Índice	Operador	Localizaciones
1	ISV	38
2	EAA	15
3	EEU	2
4	ERR	0
5	ELL	0
6	ECC	13
7	ECN	7
8	EMD	0
9	EMF	0
10	AFP	0
11	ASF	2
12	AIS	3
13	AIE	0
14	AWR	0
15	AJC	0
16	ASI	30
17	APM	0
18	APA	0
19	XMF	0
20	XMC	0
21	XMT	0
22	XTF	0
23	XER	0
24	XEE	0
25	AEL	26
26	EIU	15
27	EIN	0
28	EAP	15
29	EAN	15
30	CFA	26
31	CDE	0
32	CCO	0
33	CDC	0

Tabla 5.2: Operadores aplicados en squaresSum_2

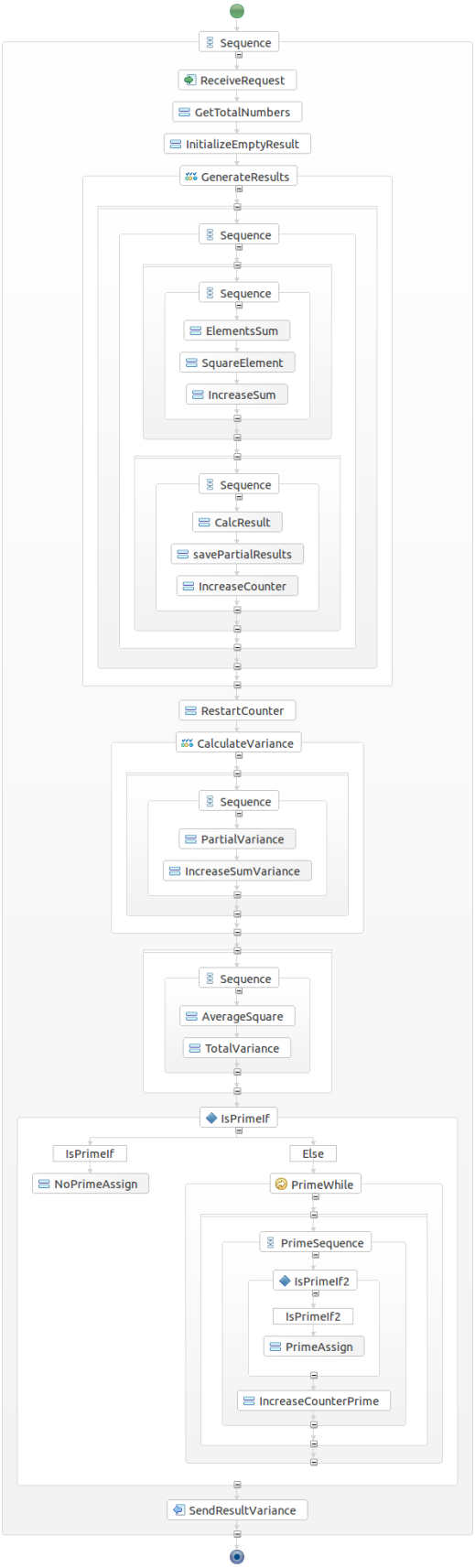

Figura 5.3: Comportamiento squaresSum_3

- OneIteration: El cliente manda un mensaje que contiene un número n = 1. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 1, que la varianza es 0.25, que el número de elementos contenidos en *elements* es 2, que el número de elementos contenidos en *varianceElements* es 2, que la suma los elementos contenidos en *elements* es 1 y que la suma de los elementos contenidos en *varianceElements* es 0.5. Además, se comprueba que el valor del elemento *prime* sea "false" ya que el número recibido en el mensaje no es primo.
- SeveralIterations: El cliente manda un mensaje que contiene un número n = 100. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 338350, que la varianza es 850, que el número de elementos contenidos en *elements* es 101, que el número de elementos contenidos en *varianceElements* es 101, que la suma los elementos contenidos en *elements* es 338350 y que la suma de los elementos contenidos en *varianceElements* es 3350.000000000001. Además, se comprueba que el valor del elemento *prime* sea "false" ya que el número recibido en el mensaje no es primo.
- FiveIterations: El cliente manda un mensaje que contiene un número n = 5. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 55, que la varianza es 2.916666666666668, que el número de elementos contenidos en *elements* es 6, que el número de elementos contenidos en *varianceElements* es 6, que la suma los elementos contenidos en *elements* es 55 y que la suma de los elementos contenidos en *varianceElements* es 9.166666666666668. Además, se comprueba que el valor del elemento *prime* sea "true" ya que el número recibido en el mensaje es primo.
- FourIterations: El cliente manda un mensaje que contiene un número n = 4. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 30, que la varianza es 2, que el número de elementos contenidos en *elements* es 5, que el número de elementos contenidos en *varianceElements* es 5, que la suma los elementos contenidos en *elements* es 30 y que la suma de los elementos contenidos en *varianceElements* es 6. Además, se comprueba que el valor del elemento *prime* sea "false" ya que el número recibido en el mensaje no es primo.
- NineIterations: El cliente manda un mensaje que contiene un número n = 9. Cuando se recibe el mensaje de respuesta se comprueba que la suma total de potencias es 285, que la varianza es 8.25, que el número de elementos contenidos en *elements* es 10, que el número de elementos contenidos en *varianceElements* es 10, que la suma los elementos contenidos en *elements* es 285 y que la suma de los elementos contenidos en *varianceElements* es 28.5. Además, se comprueba que el valor del elemento *prime* sea "false" ya que el número recibido en el mensaje es primo.

Los operadores que se aplican en esta composición se pueden observar en la tabla 5.3.

En total, con esta composición se generan 447 mutantes. De los cuales, con el conjunto de casos de pruebas empleado, 402 mueren, 2 son no válidos y 43 quedan vivos, siendo mutantes equivalentes todos los que quedan vivos. Por lo que el conjunto de casos de prueba empleado mata a todos los mutantes no equivalentes.

5.4. Composición tradeIncome

Mediante esta composición se modela el comportamiento de la gestión de un supermercado, realizando un control de los beneficios totales que se han generado hasta una determinada fecha por los distintos establecimientos que dicho supermercado posee. Se controlan, además de los beneficios, los ingresos, los costes, si la gestión ha resultado rentable, la sección en la que se han obtenido más beneficios, el stock, si es necesario realizar algún tipo de marketing (ofertas o publicidad) y si es necesario realizar el informe anual del supermercado.

Índice	Operador	Localizaciones
1	ISV	84
2	EAA	17
3	EEU	3
4	ERR	4
5	ELL	2
6	ECC	16
7	ECN	12
8	EMD	0
9	EMF	0
10	AFP	0
11	ASF	3
12	AIS	3
13	AIE	1
14	AWR	1
15	AJC	0
16	ASI	38
17	APM	0
18	APA	0
19	XMF	0
20	XMC	0
21	XMT	0
22	XTF	0
23	XER	0
24	XEE	0
25	AEL	34
26	EIU	21
27	EIN	3
28	EAP	21
29	EAN	21
30	CFA	34
31	CDE	3
32	CCO	4
33	CDC	6

Tabla 5.3: Operadores aplicados en squaresSum_3

El objetivo inicial para la realización de esta composición fue la aplicación del operador ACI, que como se explica en el apéndice B, este operador cambia el atributo *createInstance* de una actividad de recepción de mensajes de “yes” a “no”. Para ello, la composición debe cumplir con unos requisitos, es decir, tiene que haber más de una actividad de recepción de mensajes con el atributo *createInstance* a “yes” y la estructura que siga la composición debe tener en cuenta los siguientes casos:

```
<process>
  <sequence>
    <flow>
      <receive createInstance="yes" /> → se descarta
      <receive createInstance="yes" /> → se descarta
    </flow>
  </sequence>
</process>


<process>
  <sequence>
    <flow>
      <receive createInstance="yes" /> → se descarta
        <sequence>
          <receive createInstance="yes" /> → se descarta
          ...
        </sequence>
    </flow>
  </sequence>
</process>


<process>
  <sequence>
    <receive createInstance="yes"/> → se descarta
    ...
    <flow>
      <receive createInstance="yes"/> → produce mutante 1
      <receive createInstance="yes"/> → produce mutante 2
    </flow>
    ...
  </sequence>
</process>
```

Es decir, en los dos primeros casos, el operador ACI no produciría ningún mutante debido a que serían no válidos según las comprobaciones estáticas que realiza WS-BPEL. Sin embargo, en el tercer caso, donde el *flow* en el que están los dos *receive* tiene a un sequence como padre sin estar el primero, sí se deberían producir dos mutantes por el operador ACI.

Por lo tanto, la composición se desarrolló siguiendo una estructura similar a la del tercer caso para que pudiera producir mutantes válidos de ACI. Sin embargo, cuando la composición se probó, se observó que tampoco podía aplicarse debido a las restricciones que impone el motor ActiveBPEL, más estrictas aún, por lo que no permite que se pueda implementar una composición con la estructura comentada anteriormente. Lo que imposibilita que se pueda aplicar el operador ACI y, por lo tanto, se decide descartar dicho operador.

De esta manera, el objetivo en el que se centra esta composición es en la aplicación de ciertos operadores aplicados de manera insuficiente con anterioridad, como son los operadores: EEU, ELL,

ECC, EMF, AIS, AFP, AWR, AJC, APM, APA, XMF, XMC, XMT, XTF, XER y XEE. Finalmente, en esta composición se pueden aplicar todos los operadores disponibles para WS-BPEL 2.0.

Esta composición se comporta de la siguiente manera:

- La central del supermercado envía un mensaje con una determinada fecha, y cada uno de los establecimientos envía otro con la siguiente estructura:

```
earnings/
  section/
    total ← Ingresos totales de la seccion.
    costs/
      total ← Costes totales de cada parte <costs>.
      stock
```

- Se calculan los ingresos, costes, beneficios y stock total de los establecimientos. Así como cuál es la sección en la que más beneficios se obtienen.
- Se invoca al servicio rankService que proporciona una cantidad a partir de la cual se considera si hay beneficios o no.
- Cuando se han calculado todos los elementos se envía un mensaje a un servicio llamado establishmentsControl confirmando que se ha realizado con éxito.
- Se envía el mensaje de respuesta con todos los elementos calculados anteriormente.
- También se dispone de:
  - Una operación para poder abortar la ejecución del proceso.
  - Un servicio para contabilizar costes adicionales como impuestos, seguro, alquiler, etc.
  - Un servicio para considerar publicidad u ofertas para los siguientes meses.

En esta composición se describen un total de 7 servicios: ManagementService, EstablishmentService1, EstablishmentService2, RankService, EstablishmentsControlService, AccountingService y MarketingService.

- **ManagementService:** Servicio con el que se representa la entidad que gestiona el supermercado. Cuenta con una operación para abortar el proceso.
- **EstablishmentService1:** Servicio con el que se representa un establecimiento del supermercado.
- **EstablishmentService2:** Servicio con el que se representa otro establecimiento del supermercado.
- **RankService:** Servicio mediante el que se obtiene la cantidad a partir de la cual se consideran que las ganancias totales del supermercado han sido productivas.
- **EstablishmentsControlService:** Servicio encargado de controlar que las operaciones con los datos enviados por los distintos establecimientos se han realizado correctamente.
- **AccountingService:** Servicio mediante el que se contabiliza el total de los posibles costes adicionales durante la gestión del supermercado.
- **MarketingService:** Servicio a través del cuál se define si se va a realizar algún tipo de marketing, ya sean ofertas o publicidad.

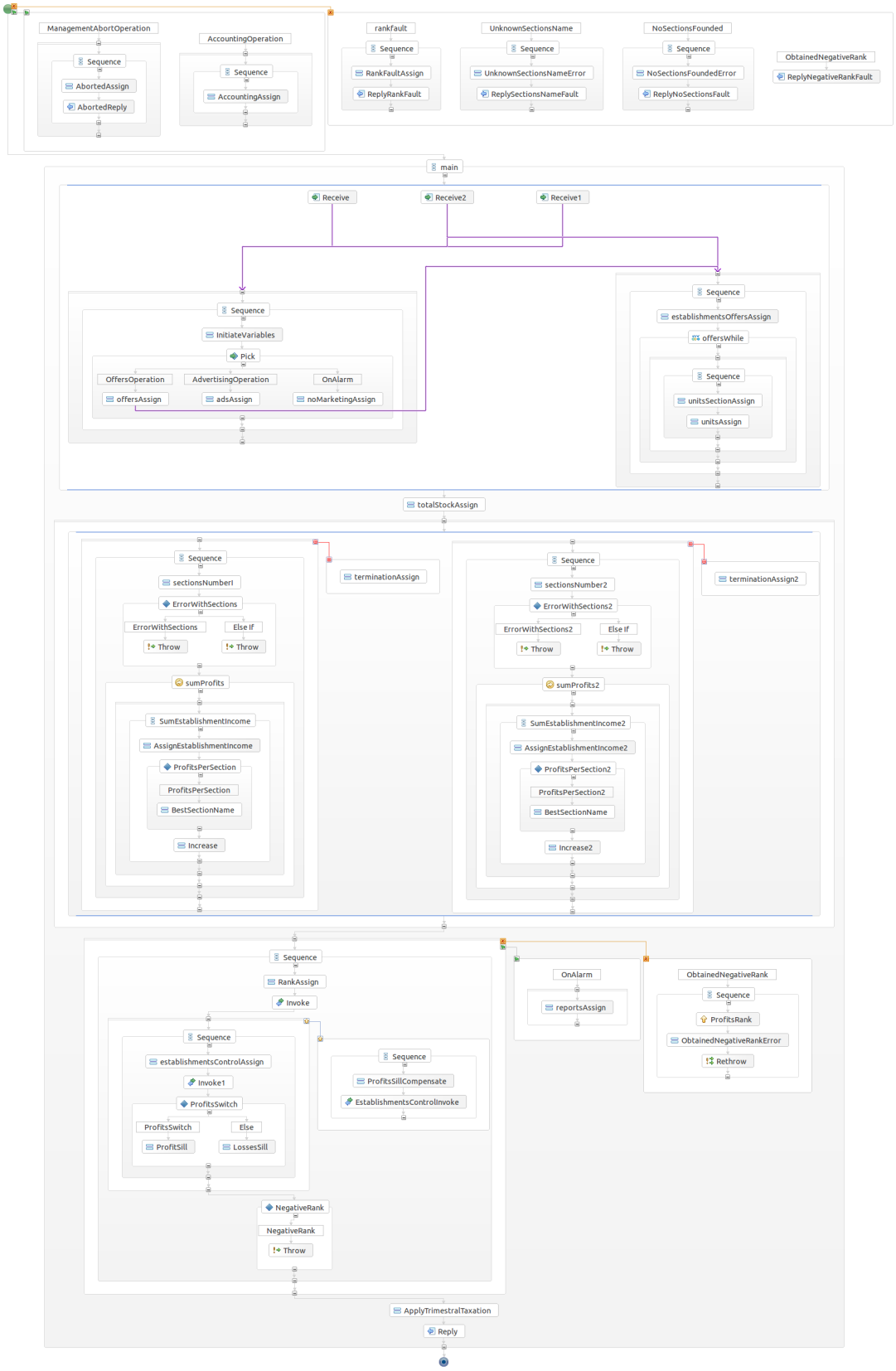

Figura 5.4: Comportamiento tradeIncome

Los casos de prueba están incluidos en un fichero bpts. Son los siguientes:

- **firstIncome:** Mediante este caso de prueba se representa una gestión del supermercado en la que el partner encargado del control de su gestión envía un mensaje con una determinada fecha, los dos establecimientos mandan otro con la misma fecha y los datos necesarios para la realización de los distintos cálculos que requieren la gestión del supermercado. Además, se envía un mensaje a través del partner encargado del marketing, definiendo el tipo de publicidad que se realizará. También se envía el mensaje que contiene la cantidad con la que se determina si los beneficios obtenidos son satisfactorios o no, mediante el partner encargado de ello. Cuando se realizan los cálculos con los datos de los establecimientos se envía un mensaje de aprobación al partner encargado de su control. Y, por último, se termina el proceso mediante el envío de un mensaje de respuesta al partner encargado del control de la gestión del supermercado.
- **secondIncome:** En este caso de prueba se sigue un comportamiento similar al anterior, con la diferencia de que el mensaje que se envía por el partner encargado del marketing define las ofertas a aplicar.
- **thirdIncome:** Este caso de prueba también sigue un comportamiento semejante al que se define en *firstIncome*. Incluido el mensaje que se envía por parte del partner encargado del marketing, definiendo el tipo de publicidad que se aplicará.
- **fourthIncome:** Caso de prueba en el que el comportamiento también es similar a los anteriores. Con la peculiaridad de que en este caso de prueba se determina que no se obtienen beneficios, produciéndose pérdidas para el supermercado. Al igual que en *firstIncome* y *thirdIncome*, se envía un mensaje por parte del partner encargado del marketing, definiendo el tipo de publicidad que se aplicará.
- **IncomeRankFaultTest:** Caso de prueba en el que el partner encargado de enviar el mensaje con la cantidad a partir de la cual se obtienen beneficios, envía un fallo en su lugar.
- **NoSectionsNameFaultTestI:** Con este caso de prueba se modela que el segundo establecimiento mande un mensaje en el que falta al menos un nombre de algunas de las secciones de las que consta dicho establecimiento.
- **NoSectionsNameFaultTestII:** Caso de prueba con el que se modela que el primer establecimiento mande un mensaje en el que falta al menos un nombre de algunas de las secciones de las que consta dicho establecimiento.
- **NoSectionsFaultTestI:** Con este caso de prueba se modela que el segundo establecimiento mande un mensaje en el que no haya ninguna sección.
- **NoSectionsFaultTestII:** Caso de prueba con el que se modela que el primer establecimiento mande un mensaje en el que no haya ninguna sección.
- **NegativeRankFaultTest:** Mediante este caso de prueba se modela que el partner encargado de enviar el mensaje con la cantidad a partir de la cual se obtienen beneficios, envíe una cantidad que sea negativa.
- **AbortFaultTest:** Con este caso de prueba se modela que el partner encargado de controlar la gestión del supermercado envíe un mensaje con el que se aborte la ejecución del proceso.
- **TaxationEventTest:** Mediante este caso de prueba se modela que envíe un mensaje el partner mediante el que se contabiliza el total de los posibles costes adicionales durante la gestión del supermercado.

A continuación, podemos observar el código correspondiente al caso de prueba *firstIncome*:

En primer lugar encontramos las actividades que realiza el socio de negocio que actúa como cliente, encargado del control de la gestión del supermercado. Manda un mensaje que consta de una fecha determinada y recibe otro mensaje para el que tiene una serie de condiciones que deben cumplirse.

```
<tes:testCase name="firstIncome" basedOn="" abstract="false" vary="false">
 <tes:clientTrack>
  <tes:sendReceive service="trad:ManagementService" port="ManagementPort"
      operation="ManagementOperation">
   <tes:send service="trad:ManagementService" port="ManagementPort"
       operation="ManagementOperation" fault="false">
    <tes:data>
     <trad:ManagementOperation>
      <requestDate>2012-02-02</requestDate>
     </trad:ManagementOperation>
    </tes:data>
   </tes:send>
   <tes:receive service="trad:ManagementService" port="ManagementPort"
       operation="ManagementOperation" fault="false">
    <tes:condition>
     <tes:expression>trad:ManagementOperationResponse/profits</
         tes:expression>
     <tes:value>15000</tes:value>
    </tes:condition>
    <tes:condition>
     <tes:expression>trad:ManagementOperationResponse/income</
         tes:expression>
     <tes:value>30400</tes:value>
    </tes:condition>
    <tes:condition>
     <tes:expression>trad:ManagementOperationResponse/costs</
         tes:expression>
     <tes:value>15400</tes:value>
    </tes:condition>
    <tes:condition>
     <tes:expression>trad:ManagementOperationResponse/profitPass</
         tes:expression>
     <tes:value>'Profitable'</tes:value>
    </tes:condition>
    <tes:condition>
     <tes:expression>trad:ManagementOperationResponse/bestSection/name</
         tes:expression>
     <tes:value>'fishMarket'</tes:value>
    </tes:condition>
    <tes:condition>
     <tes:expression>trad:ManagementOperationResponse/bestSection/
         establishment</tes:expression>
     <tes:value>1</tes:value>
    </tes:condition>
    <tes:condition>
     <tes:expression>trad:ManagementOperationResponse/bestSection/profits<
         /tes:expression>
     <tes:value>4300</tes:value>
    </tes:condition>
    <tes:condition>
```

```
41        <tes:expression>trad:ManagementOperationResponse/totalStock</
              tes:expression>
42        <tes:value>2055</tes:value>
43       </tes:condition>
44       <tes:condition>
45        <tes:expression>trad:ManagementOperationResponse/marketing</
              tes:expression>
46        <tes:value>'Ads will be make the next month'</tes:value>
47       </tes:condition>
48       <tes:condition>
49        <tes:expression>trad:ManagementOperationResponse/deals/sectionsUnits<
              /tes:expression>
50        <tes:value>0</tes:value>
51       </tes:condition>
52       <tes:condition>
53        <tes:expression>trad:ManagementOperationResponse/reports</
              tes:expression>
54        <tes:value>'none'</tes:value>
55       </tes:condition>
56      </tes:receive>
57     </tes:sendReceive>
58    </tes:clientTrack>
```

A continuación, nos encontramos con el mensaje que envía cada uno de los dos supermercados.

```
1     <tes:partnerTrack name="Establishment1">
2      <tes:sendOnly service="trad:Establishment1Service" port="
           Establishment1Port" operation="EstablishmentOperation" fault="false"
           delay="1">
3       <tes:data>
4        <trad:EstablishmentOperation>
5         <collectionDate>2012-02-02</collectionDate>
6         <earnings>
7          <section name="textil">
8           <total>2450</total>
9           <costs name="t-shirts">
10           <total>775</total>
11           <stock>50</stock>
12          </costs>
13          <costs name="pants">
14           <total>225</total>
15           <stock>60</stock>
16          </costs>
17         </section>
18         <section name="fishMarket">
19          <total>7000</total>
20          <costs name="blue">
21           <total>1300</total>
22           <stock>250</stock>
23          </costs>
24          <costs name="white">
25           <total>1400</total>
26           <stock>300</stock>
27          </costs>
28         </section>
```

```
29          <section name="butchery">
30           <total>5000</total>
31           <costs name="pork">
32            <total>1200</total>
33            <stock>300</stock>
34           </costs>
35           <costs name="veal">
36            <total>1000</total>
37            <stock>250</stock>
38           </costs>
39           <costs name="sausages">
40            <total>500</total>
41            <stock>100</stock>
42           </costs>
43          </section>
44         </earnings>
45        </trad:EstablishmentOperation>
46       </tes:data>
47      </tes:sendOnly>
48     </tes:partnerTrack>
49     <tes:partnerTrack name="Establishment2">
50      <tes:sendOnly service="trad:Establishment2Service" port="
           Establishment2Port" operation="EstablishmentOperation" fault="false"
           delay="2">
51       <tes:data>
52        <trad:EstablishmentOperation>
53         <collectionDate>2012-02-02</collectionDate>
54         <earnings>
55          <section name="greengrocers">
56           <total>2500</total>
57           <costs name="fruits">
58            <total>700</total>
59            <stock>150</stock>
60           </costs>
61           <costs name="vegetables">
62            <total>600</total>
63            <stock>125</stock>
64           </costs>
65          </section>
66          <section name="multimedia">
67           <total>9000</total>
68           <costs name="computers">
69            <total>2000</total>
70            <stock>50</stock>
71           </costs>
72           <costs name="screens">
73            <total>1800</total>
74            <stock>30</stock>
75           </costs>
76           <costs name="tables">
77            <total>2100</total>
78            <stock>80</stock>
79           </costs>
80          </section>
81          <section name="textil">
82           <total>2000</total>
```

```
          <costs name="t-shirts">
           <total>300</total>
           <stock>90</stock>
          </costs>
          <costs name="pants">
           <total>500</total>
           <stock>110</stock>
          </costs>
         </section>
         <section name="bakery">
          <total>2450</total>
          <costs name="bread">
           <total>775</total>
           <stock>50</stock>
          </costs>
          <costs name="cakes">
           <total>225</total>
           <stock>60</stock>
          </costs>
         </section>
        </earnings>
       </trad:EstablishmentOperation>
      </tes:data>
     </tes:sendOnly>
    </tes:partnerTrack>
```

El siguiente socio es el encargado de enviar la cantidad a partir de la cual se considera si la gestión ha sido provechosa o no. Recibe un mensaje con una determinada fecha y envía otro con una determinada cantidad.

```
<tes:partnerTrack name="IncomeRank">
 <tes:receiveSend service="trad:RankService" port="RankPort" operation="
    RankOperation">
  <tes:receive service="trad:RankService" port="RankPort" operation="
     RankOperation" fault="false">
   <tes:condition>
    <tes:expression>trad:RankOperation/requestDate</tes:expression>
    <tes:value>'2012-02-02'</tes:value>
   </tes:condition>
  </tes:receive>
  <tes:send service="trad:RankService" port="RankPort" operation="
     RankOperation" fault="false">
   <tes:data>
    <trad:RankOperationResponse>
     <profitsAmount>12000</profitsAmount>
    </trad:RankOperationResponse>
   </tes:data>
  </tes:send>
 </tes:receiveSend>
</tes:partnerTrack>
```

La actividad que realiza el siguiente socio es la del control de que los datos enviados por los establecimientos se han utilizado para la gestión del supermercado correctamente. Recibe un mensaje, para los que tiene una serie de condiciones que deben cumplirse.

```
<tes:partnerTrack name="tradeControl">
 <tes:receiveOnly service="trad:EstablishmentsControlService" port="
    EstablishmentsControlPort" operation="EstablishmentsControlOperation"
     fault="false">
  <tes:condition>
   <tes:expression>trad:EstablishmentsControlOperation/controlDate</
      tes:expression>
   <tes:value>'2012-02-02'</tes:value>
  </tes:condition>
  <tes:condition>
   <tes:expression>trad:EstablishmentsControlOperation/controlInformation
      </tes:expression>
   <tes:value>'The information of the establishments have been processed
      correctly'</tes:value>
  </tes:condition>
 </tes:receiveOnly>
</tes:partnerTrack>
```

Y por último, el socio encargado del marketing envía un mensaje con una determinada fecha y el tipo de publicidad que se debe emplear.

```
<tes:partnerTrack name="marketingTrade">
 <tes:sendOnly service="trad:MarketingService" port="MarketingPort"
    operation="AdvertisingOperation" fault="false">
  <tes:data>
   <trad:AdvertisingOperation>
    <advertisingDate>2012-02-02</advertisingDate>
    <advertising>
     <ad>
      <media>TV</media>
      <cost>2500</cost>
     </ad>
    </advertising>
   </trad:AdvertisingOperation>
  </tes:data>
 </tes:sendOnly>
</tes:partnerTrack>
</tes:testCase>
```

Los operadores que se aplican en esta composición se pueden observar en la tabla 5.4.

En total, con esta composición se generan 815 mutantes. De los cuales, con el conjunto de casos de pruebas empleado, 668 mueren, 19 son no válidos y 128 quedan vivos, siendo mutantes equivalentes todos los que quedan vivos. Por lo que el conjunto de casos de prueba empleado mata a todos los mutantes no equivalentes.

Índice	Operador	Localizaciones
1	ISV	126
2	EAA	14
3	EEU	2
4	ERR	12
5	ELL	2
6	ECC	130
7	ECN	7
8	EMD	1
9	EMF	1
10	AFP	1
11	ASF	13
12	AIS	2
13	AIE	3
14	AWR	2
15	AJC	1
16	ASI	47
17	APM	2
18	APA	2
19	XMF	5
20	XMC	1
21	XMT	2
22	XTF	4
23	XER	1
24	XEE	2
25	AEL	88
26	EIU	19
27	EIN	11
28	EAP	19
29	EAN	19
30	CFA	88
31	CDE	11
32	CCO	13
33	CDC	14

Tabla 5.4: Operadores aplicados en tradeIncome

Capítulo 6

Conclusiones y trabajo futuro

El desarrollo de este PFC ha sido de gran utilidad para mí, puesto que he conseguido obtener bastantes conocimientos nuevos. Además, se trata de la primera vez que colaboro con un grupo de investigación y ha resultado ser una experiencia bastante grata, en la que he podido aprender como se trabaja en grupo asistiendo a distintos seminarios durante el desarrollo de este PFC, donde se podían plantear todas las dudas y problemas que fueran surgiendo a lo largo del proyecto.

El propósito de este proyecto estaba enfocado hacia un tema totalmente nuevo para mí, por lo que tuve que realizar una fase de aprendizaje que me sería de gran ayuda a lo largo del desarrollo de este proyecto. Dentro de los conocimientos adquiridos se encuentra la prueba de mutaciones, técnica hacia la que ha sido dirigida el desarrollo de las distintas composiciones WS-BPEL. Así, como el comportamiento de los distintos operadores de mutación de los que se dispone para WS-BPEL.

También se ha aprendido a programar con el lenguaje WS-BPEL, un lenguaje que desconocía por completo, con muy poco en común con los lenguajes que se suelen estudiar en la Ingeniería Técnica y que, además, resulta algo complicado tanto de aprender como de poner en práctica. Igualmente, ha sido necesario adquirir conocimientos sobre el resto de tecnologías y lenguajes necesarios para el desarrollo de las distintas composiciones, como WSDL, BPELUnit, XPath, XML Schema, SOAP, TestSPec y Velocity.

Asimismo, gracias a este proyecto, he podido practicar y llegar a desenvolverme de una manera fluida con la herramienta ECLIPSE, utilizada para el desarrollo de las composiciones. También he podido aprender a utilizar diferentes herramientas necesarias para el desarrollo de este proyecto, como XMLEye, TkDiff, MuBPEL, además del sistema de control de versiones Subversion.

Por otro lado, también se trata de la primera vez que desarrollo una memoria para un proyecto de estas dimensiones. Experiencia de la que también he aprendido, y he podido practicar con las herramientas necesarias para el desarrollo de este documento, como DIA, con la que he podido realizar los diagramas que se han utilizado en este proyecto, o GranttProject, utilizada para realizar el diagrama de Gantt correspondiente. Además, para el desarrollo de esta memoria se ha utilizado el lenguaje LaTeX, que resulta bastante cómodo para escribir documentos de gran tamaño, y del que sólo tenía algunas nociones antes de realizar este proyecto, por lo que también me ha servido para profundizar en el aprendizaje de dicho lenguaje.

Cabe destacar que gracias a la realización de este PFC he realizado también mejoras en mi autoaprendizaje y comprensión, puesto que la mayoría de técnicas, lenguajes y distintas tecnologías han sido estudiadas a través de las especificaciones de los distintos lenguajes, artículos y mediante su puesta en práctica. Además de contar también con la ayuda de mi tutor, por lo que también es preciso recalcar que la interacción con el tutor y el resto del grupo de investigación resulta muy importante durante el desarrollo del proyecto.

Por último, señalar que se han cumplido todos los objetivos que se pretendían en este proyecto. Tanto los relacionados con el aprendizaje de los nuevos conceptos adquiridos y de los lenguajes de pro-

gramación necesarios para el desarrollo de este proyecto, como los que se pretendían conseguir con la realización de este PFC. En este sentido, se ha realizado la elección de un editor gráfico para el desarrollo de las composiciones y se ha conseguido que todos los operadores de mutación para WS-BPEL que necesitaban ejercitarse, se apliquen en las composiciones desarrolladas. Además, dichas composiciones constan de todos los ficheros necesarios, y por lo tanto, cumplen todos los requisitos para formar parte del repositorio de composiciones WS-BPEL que gestiona el grupo UCASE de la Universidad de Cádiz, donde además cada composición consta de un artículo wiki. Por lo tanto, el grupo de investigación UCASE dispone ahora de una serie de composiciones WS-BPEL con las que poder evaluar los operadores de mutación disponibles.

Como trabajo futuro se plantea la ampliación de la composión tradeIncome. Como se ha comentado anteriormente esta composición ya aplica todos los operadores de mutación definidos para WS-BPEL, pero el grupo UCASE está interesado actualmente en medir la calidad de estos operadores y para ello necesita que todos ellos produzcan un número muy elevado de mutantes. Algunos de los operadores de mutación definidos para WS-BPEL son muy específicos y producen pocos mutantes por cada localización en la que se aplican por lo que sería deseable disponer de una composición mucho más amplia.

Capítulo 7

Resumen

Este PFC se realiza a través de un ofrecimiento del grupo de investigación UCASE, perteneciente al Departamento de Ingeniería Informática de la Universidad de Cádiz.

El proyecto se orienta dentro de una de las líneas de investigación que lleva dicho grupo. Se centra en la aplicación de la técnica de prueba de mutaciones con el lenguaje WS-BPEL 2.0, con el que se pueden crear procesos de negocio a partir de servicios web preexistentes. El principal objetivo que se pretende con este proyecto es el desarrollo de composiciones de servicios web en WS-BPEL 2.0 orientadas a la aplicación de la prueba de mutaciones, es decir, la realización de composiciones WS-BPEL adaptadas especialmente para la aplicación de los operadores de mutación definidos para tal lenguaje.

Por lo tanto, para la realización de este proyecto ha sido necesario la adquisición de una serie de conocimientos nuevos. Como la propia prueba de mutaciones, técnica de prueba software que consiste en la introducción de pequeños cambios sintácticos en un programa a probar mediante los operadores de mutación, obteniéndose así, nuevos programas con un pequeño cambio respecto al original, denominados mutantes. Estos mutantes se ejecutan contra unos casos de pruebas y se comparan las salidas que se obtienen con la salida que produce el programa original. De esta manera, podemos tanto probar el programa, como medir la calidad del conjunto de casos de pruebas empleado, pudiendo además mejorar la calidad de dicho conjunto si se considera oportuno.

Asimismo, ha sido necesario adquirir conocimientos sobre el Lenguaje WS-BPEL, con el que especificamos el comportamiento de los procesos de negocios. Así como del lenguaje WSDL, utilizado para describir los servicios que interactúan con el proceso de negocio. Del mismo modo, para la realización de los casos de pruebas ha sido empleado BPELUnit, así como Velocity, utilizado para los casos de pruebas basados en plantillas, y el lenguaje TestSpec, con el que se generan aleatoriamente los datos a utilizar en los casos de pruebas basados en plantillas. Para la realización de este proyecto también han sido necesarios otros lenguajes como XML Schema, empleado para definir la estructura de los mensajes con los que interactúan los distintos servicios, SOAP, utilizado conjuntamente con WSDL, y XPath 2.0, que permite realizar consultas a través de expresiones ubicadas en el mismo código WS-BPEL.

Además, se ha realizado la elección de un editor gráfico que facilite el desarrollo de las distintas composiciones WS-BPEL.

Cada composición de servicios web en WS-BPEL 2.0 consta de una serie de ficheros necesarios para el desarrollo de la composición:

- Un fichero con extensión *bpel* que define el proceso de negocio.
- Al menos un fichero *wsdl* donde se especifiquen los distintos servicios que van a interactuar con el proceso BPEL.
- Un fichero con extensión *bpts* donde se definan los casos de prueba contra los que ejecutar la composición y los mutantes generados a partir de ésta.

- Además, consta de un fichero con extensión *bpts* que contiene casos de prueba basados en plantillas Velocity, de forma que se consigue una automatización de la generación de casos de prueba.
- Un fichero con extensión *spec* con el que se puede generar aleatoriamente los datos de prueba que se van a utilizar en los casos de pruebas basados en plantillas para una determinada composición.
- Un fichero con extensión *vm* que contiene los datos generados a partir del fichero *spec* para ser utilizados por los casos de pruebas basados en plantillas.

En este PFC se han desarrollado cuatro composiciones:

- **squaresSum_1:** Composición que, a partir de un número entero "n" dado, calcula la potencia al cuadrado de cada número entero comprendido entre 0 y "n" y la suma total de las potencias.
- **squaresSum_2:** Mediante esta composición, a partir de un número entero "n" dado, además de calcular la potencia al cuadrado de cada número entero comprendido entre 0 y "n" y la suma total de las potencias, se calcula la varianza a través de todos los números enteros comprendidos entre 0 y "n", así como el resultado de x^2/n para los mismos.
- **tradeIncome:** Composición con la que se modela el comportamiento de la gestión de un supermercado. Se realiza un balance del supermercado a través de la recaudación conseguida hasta una determinada fecha por los establecimientos de los que dispone el supermercado.
- **squaresSum_3:** Esta composición, además de realizar el mismo comportamiento que squaresSum_2, comprueba si el número entero dado es primo o no.

La herramienta MuBPEL, desarrollada por el grupo de investigación UCASE de la Universidad de Cádiz, es la empleada para la ejecución de las composiciones WS-BPEL frente a los casos de pruebas. Así como para la generación de mutantes, la comparación de las salidas de los mutantes con la de la composición original, el análisis de una composición WS-BPEL en cuanto a los operadores de mutación que se pueden aplicar en ella y la normalización de una composición WS-BPEL.

También se emplean otras herramientas necesarias para el desarrollo de este proyecto, como XMLEye, que permite visualizar las salidas producidas por las composiciones, y TkDiff, con el que podemos observar las diferencias entre una composición y un mutante generado a partir de ella.

Apéndice A

Desarrollo de composiciones con ECLIPSE

En esta sección del documento se explica como realizar una composición WS-BPEL con ECLIPSE. Se detalla como desarrollar con dicha herramienta los ficheros *wsdl*, *bpel* y *bpts*.

Para desarrollar una nueva composición con ECLIPSE, en primer lugar tenemos que crear un nuevo proyecto. Para ello tenemos que seleccionar: *File* → *New* → *Project*. A continuación, en el asistente que aparece, elegimos *Project* en la pestaña *General*, como podemos ver en la figura A.1, y le damos un nombre al nuevo proyecto (Figura A.2).

A.1. Creación de fichero *wsdl*

El siguiente paso a realizar es la creación de un nuevo fichero *wsdl*. Para ello basta con pulsar con el botón derecho del ratón encima del nuevo proyecto y seleccionamos: *New* → *Other*. Aparece de nuevo un asistente, como podemos observar en la figura A.3, en el que elegiremos *WSDL File* en la pestaña *Web Services*.

A continuación, vamos a la siguiente pantalla del asistente y le asignamos un nombre al fichero *wsdl* (Figura A.4).

Por último, como podemos ver en la figura A.5, determinamos en el asistente todos los atributos necesarios del fichero *wsdl*.

Con esto conseguimos tener ya en nuestro proyecto un fichero *wsdl* ya preparado con el que trabajar. Como podemos observar en la figura A.6, podemos definir los distintos servicios que utilizará la composición, así como los *ports*, *portTypes*, *binding*,operaciones y los mensajes que se utilizan en las operaciones

A.2. Creación de fichero *bpel*

A continuación, tenemos que incluir en nuestro proyecto un nuevo fichero *bpel*. Para ello, al igual que hicimos para añadir el fichero *wsdl*, tenemos que pulsar con el botón derecho del ratón encima del proyecto y seleccionar: *New* → *Other*. De esta manera, aparece de nuevo un asistente donde debemos elegir *BPEL Process File 2.0* en la pestaña *BPEL 2.0*, como podemos ver en la figura A.7.

Seguidamente, establecemos el modo de creación, nombre y espacio de nombres del nuevo fichero *bpel* (Figura A.8).

Posteriormente, como podemos observar en la figura A.9, seleccionamos la opción *Empty BPEL Process* en el apartado *Template*.

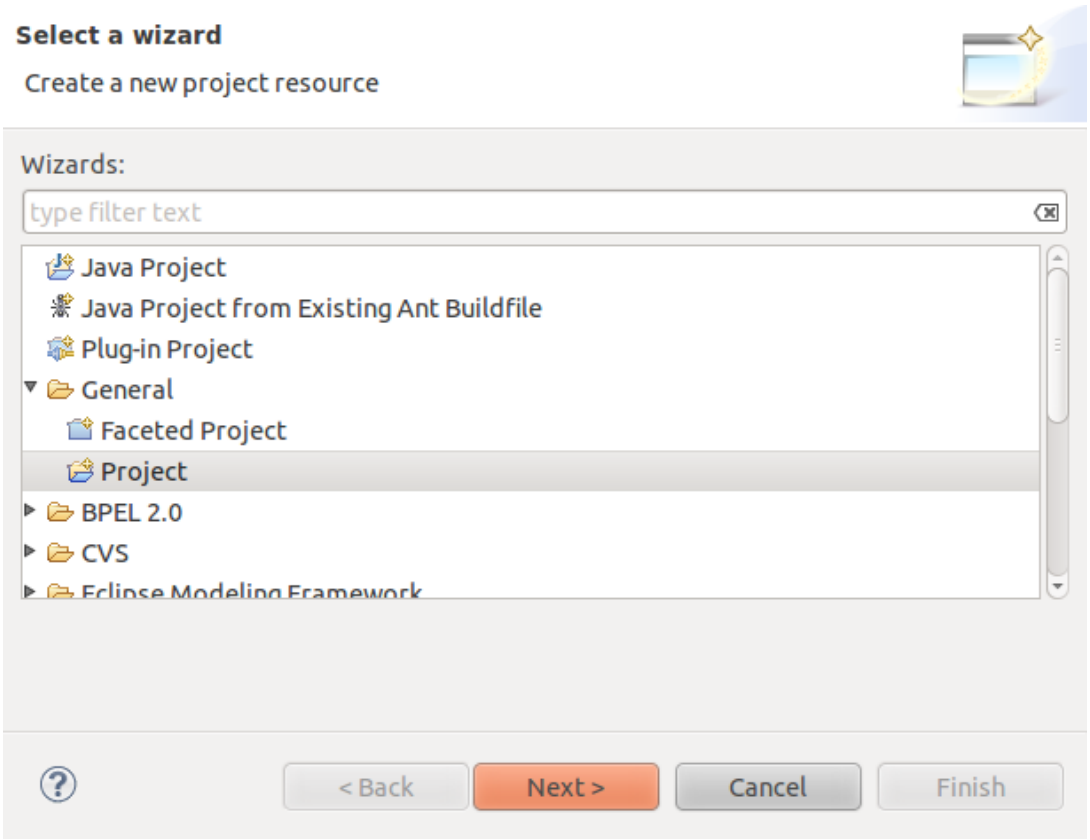

Figura A.1: Nuevo proyecto de ECLIPSE

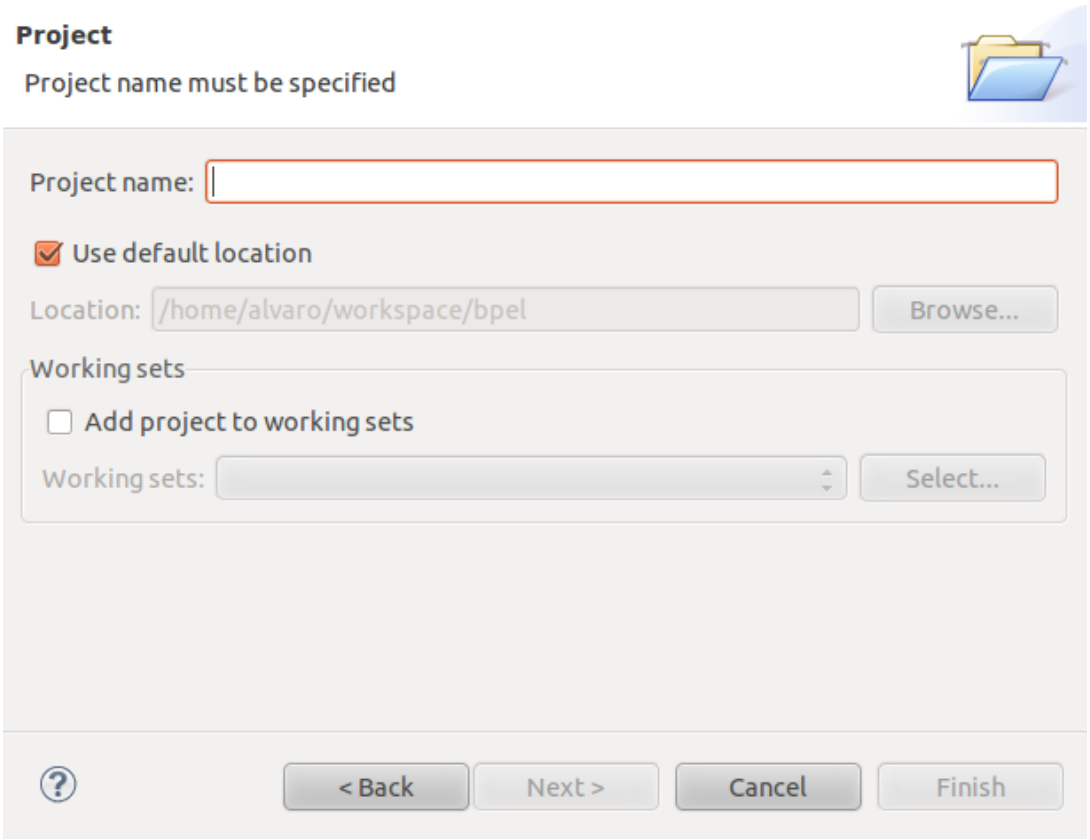

Figura A.2: Propiedades proyecto ECLIPSE

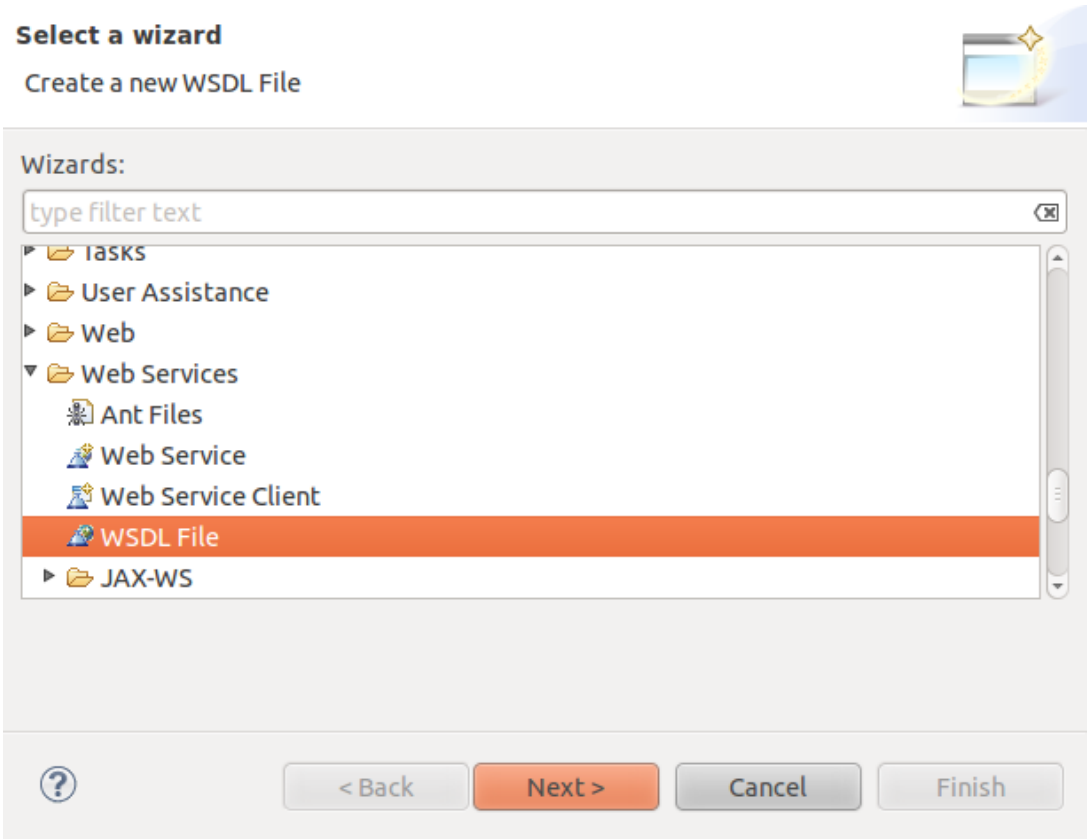

Figura A.3: Nuevo fichero WSDL

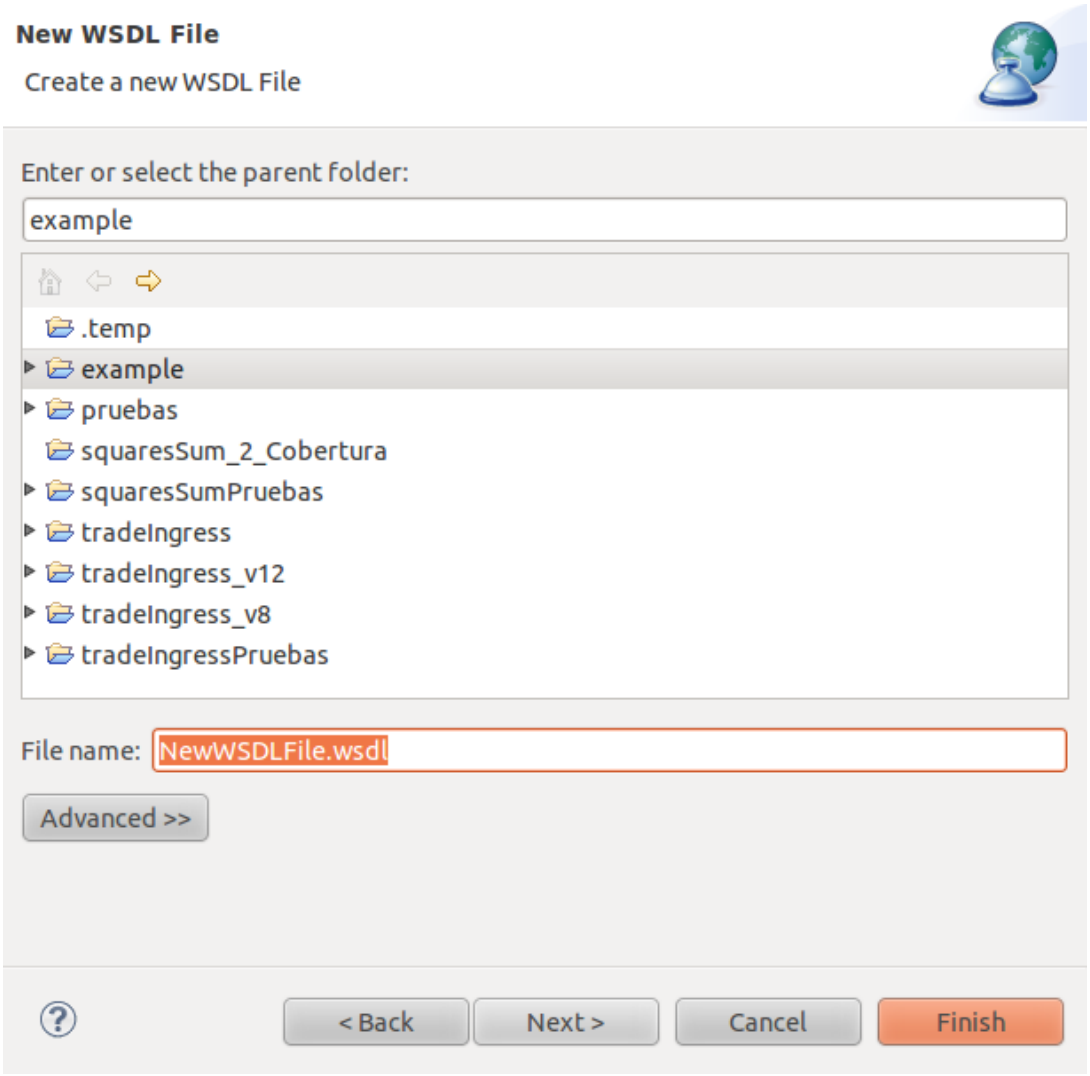

Figura A.4: Nombre fichero WSDL

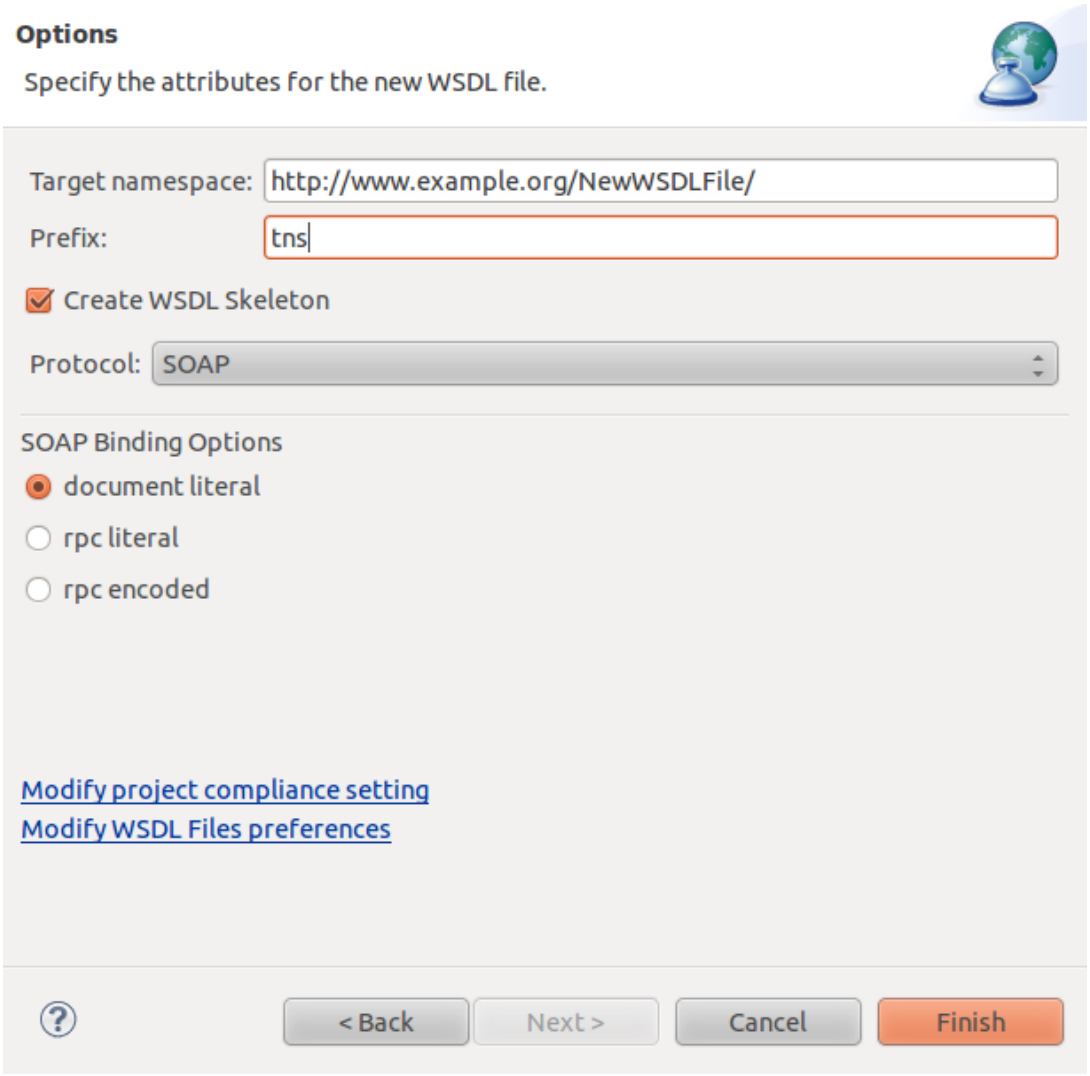

Figura A.5: Atributos fichero WSDL

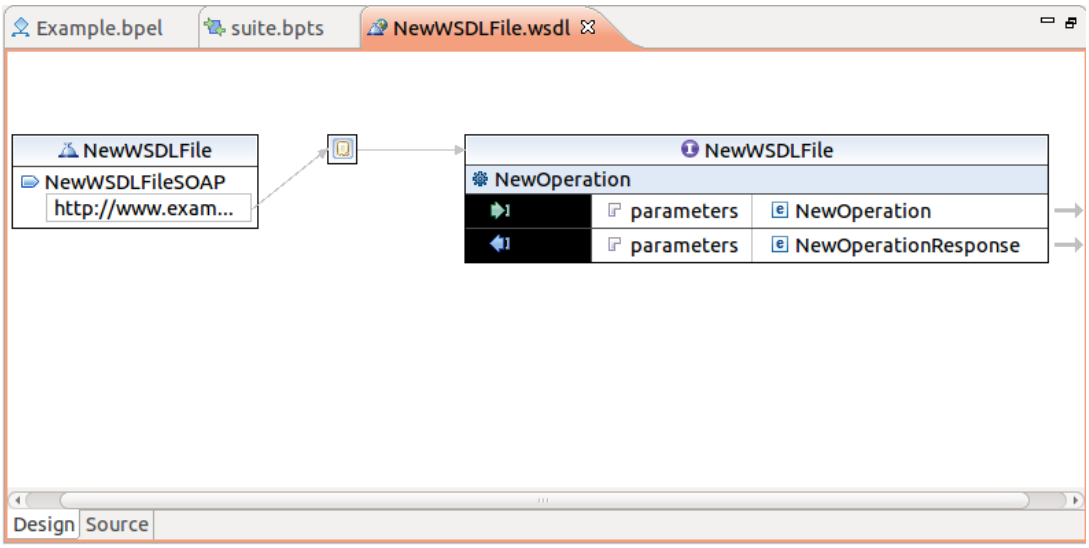

Figura A.6: Fichero WSDL

Y el último paso para tener listo el nuevo fichero *bpel*, y poder trabajar con él, es el de asignarle un nombre, como se observa en la figura A.10.

De esta manera, tenemos ya en nuestro proyecto un fichero *bpel* ya preparado con el que trabajar. Como podemos observar en la figura A.11 podemos añadir actividades en el fichero *bpel* arrastrando actividades desde la paleta al fichero.

A.3. Creación de fichero *bpts*

Para poder ejecutar la composición, además de los ficheros *wsdl* y *bpel* correspondientes, se necesita un fichero *bpts* que contenga los casos de prueba contra los que ejecutar la composición.

Para añadir un fichero *bpts* a nuestro proyecto, procedemos de igual manera que hemos hecho anteriormente con los ficheros *wsdl* y *bpel*. Pulsamos con el botón derecho del ratón encima del proyecto y seleccionamos: *New* → *Other*. De esta manera, aparece de nuevo un asistente donde debemos elegir *BPELUnit TestSuite* en la pestaña *BPELUnit* (Figura A.12).

Seguidamente, le asignamos un nombre al nuevo fichero *bpts*, como se puede observar en la figura A.13.

De este modo, como podemos ver en la figura A.14, conseguimos tener en nuestro proyecto un nuevo fichero *bpts* ya listo con el que trabajar. Y disponemos una interfaz con la que podemos añadir los distintos casos de pruebas.

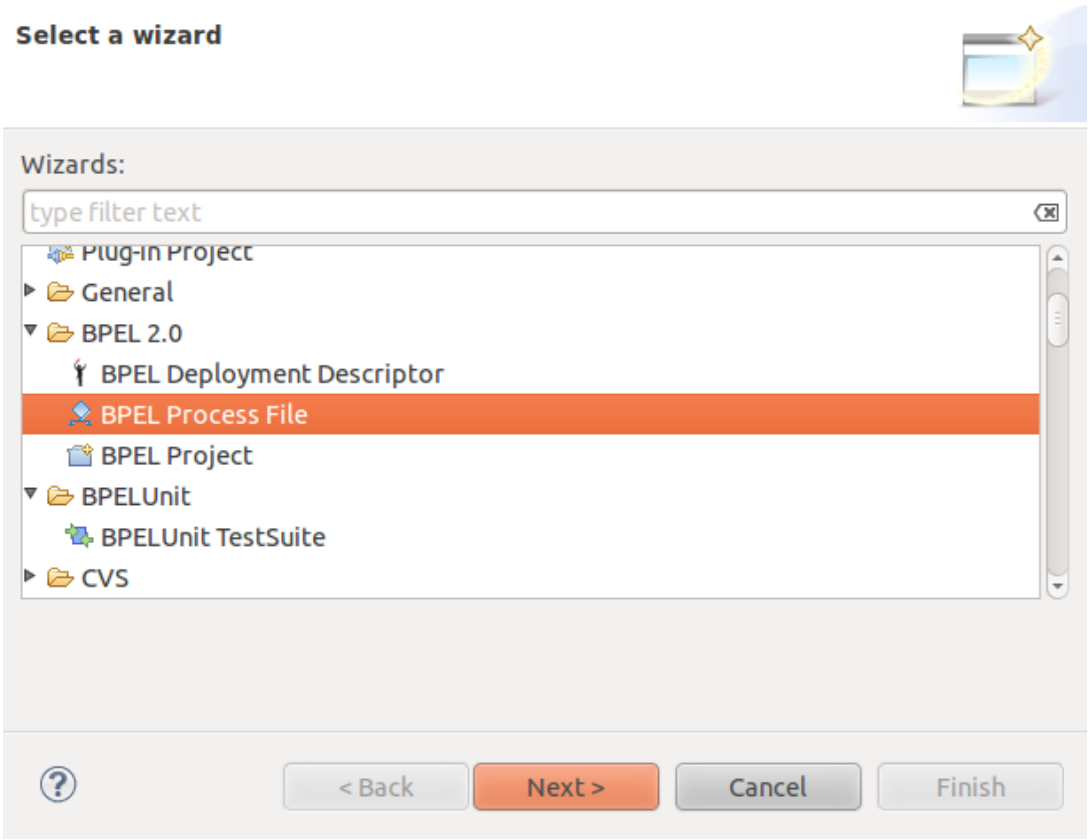

Figura A.7: Nuevo fichero BPEL

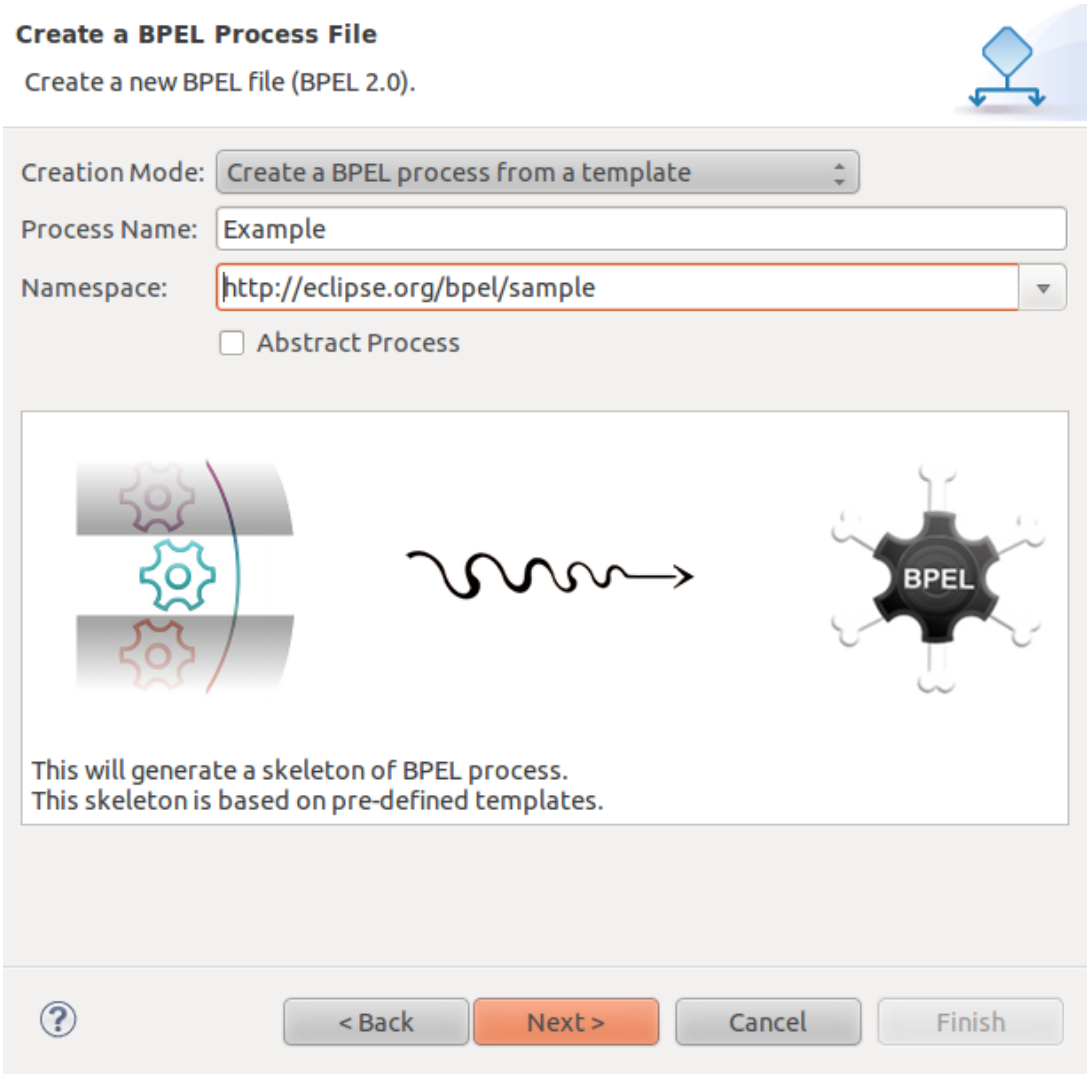

Figura A.8: Propiedades fichero BPEL

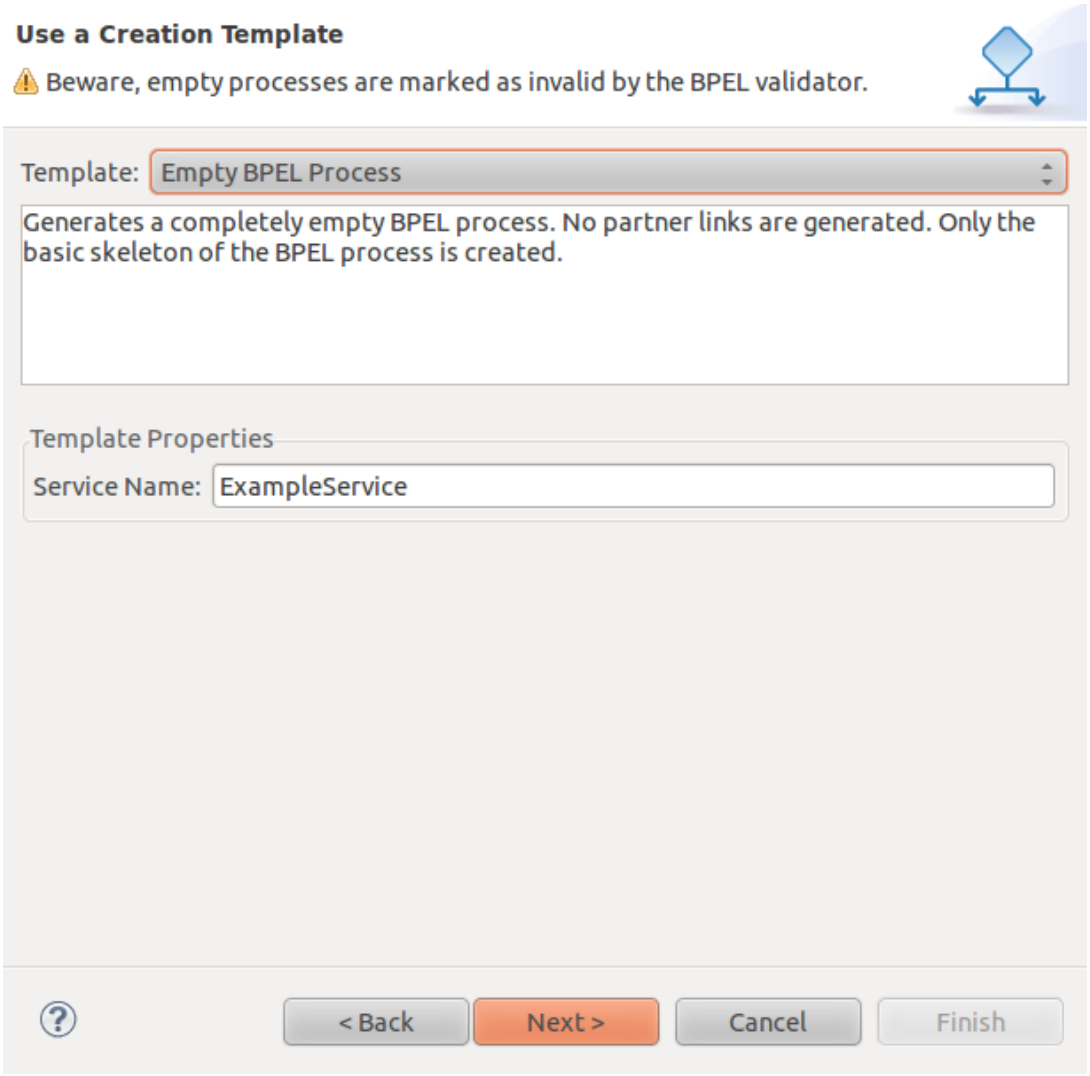

Figura A.9: Tipo fichero BPEL

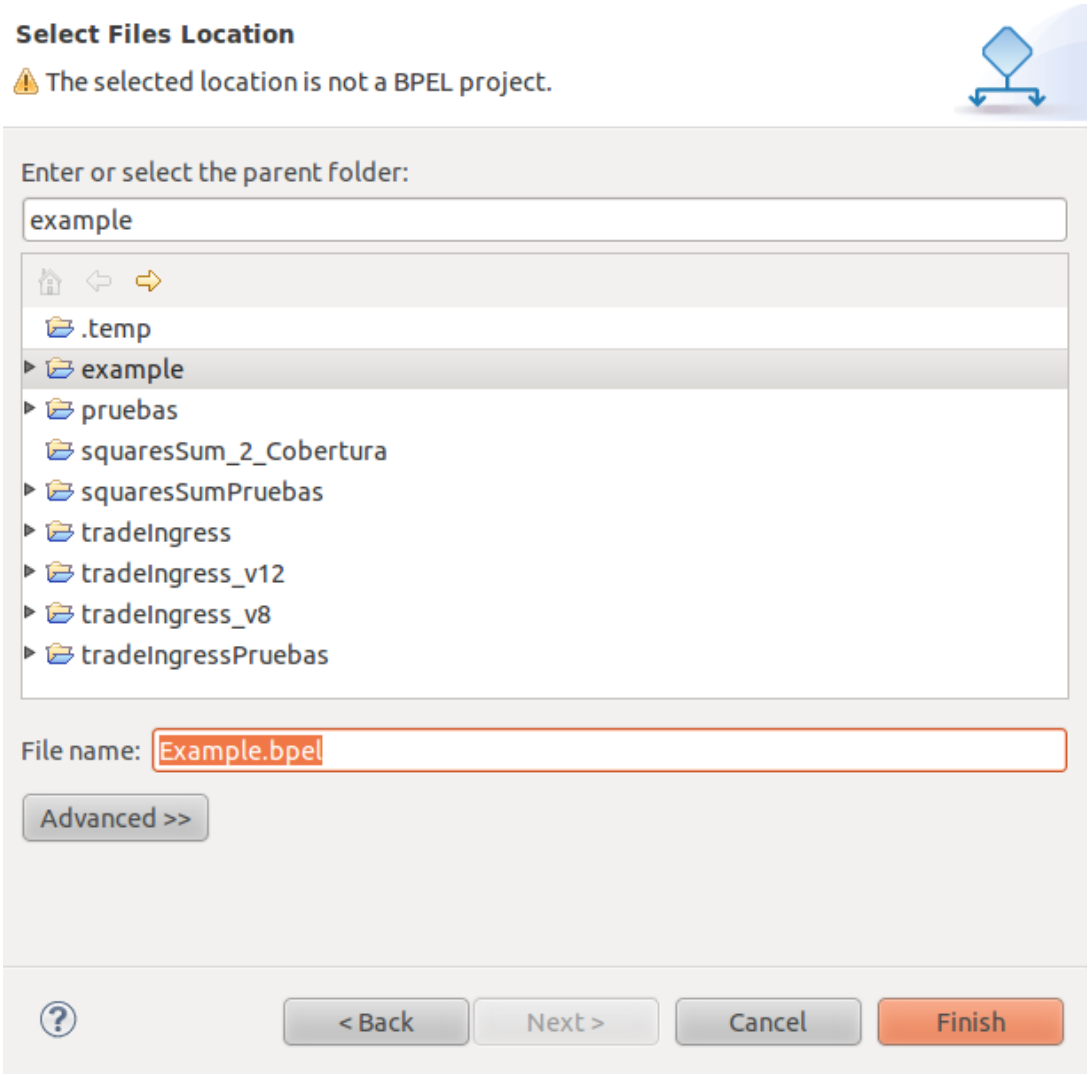

Figura A.10: Nombre fichero BPEL

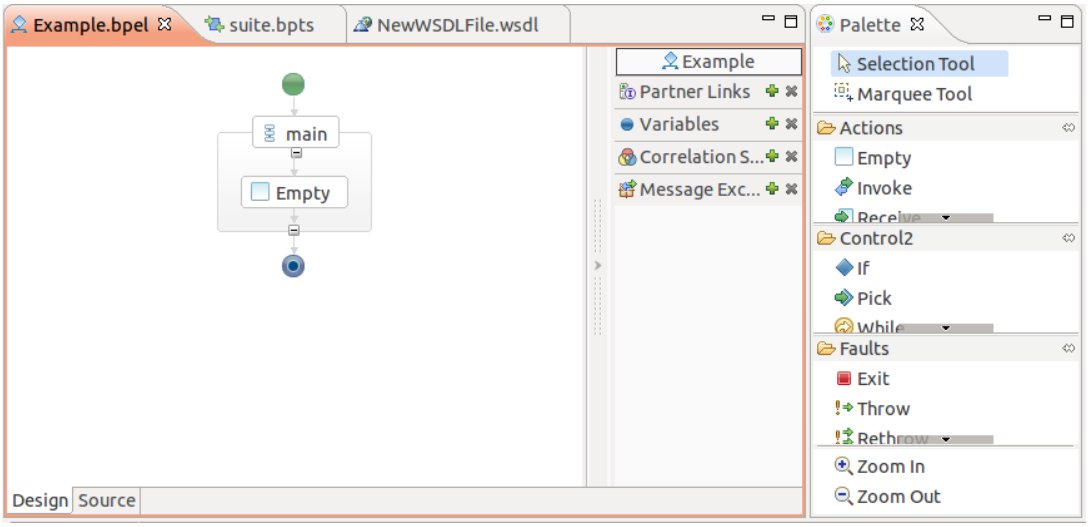

Figura A.11: Fichero BPEL

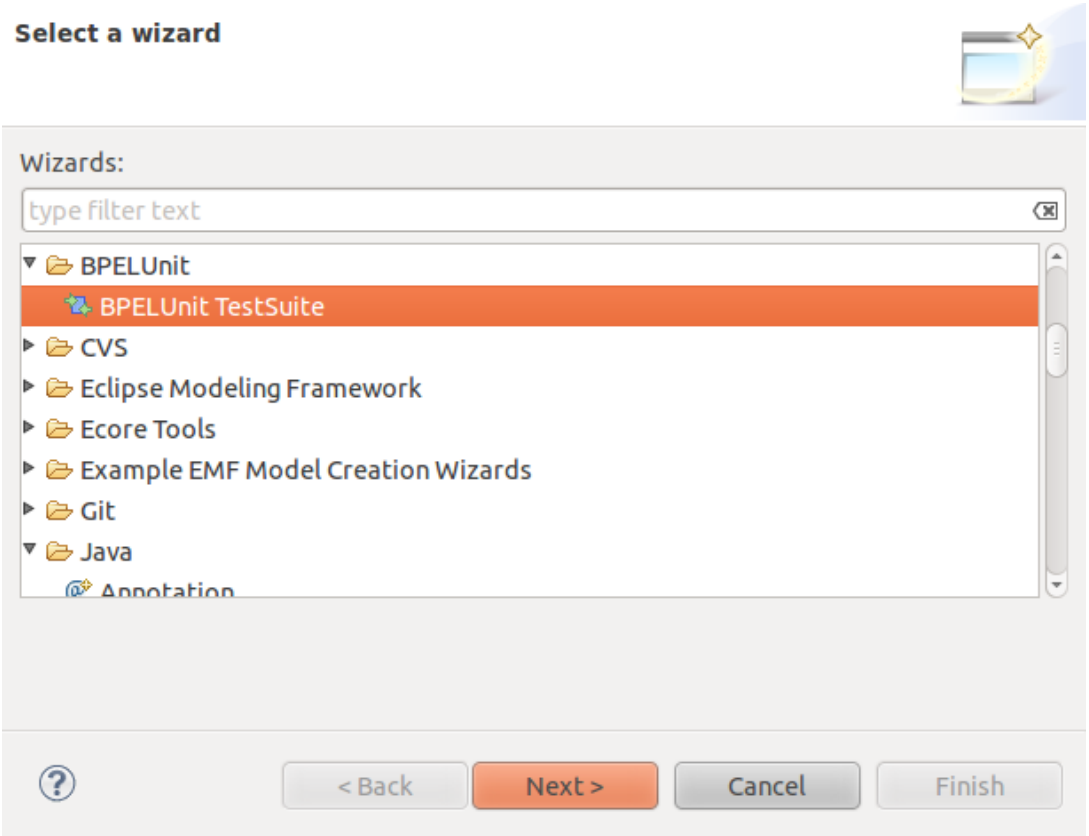

Figura A.12: Nuevo fichero BPTS

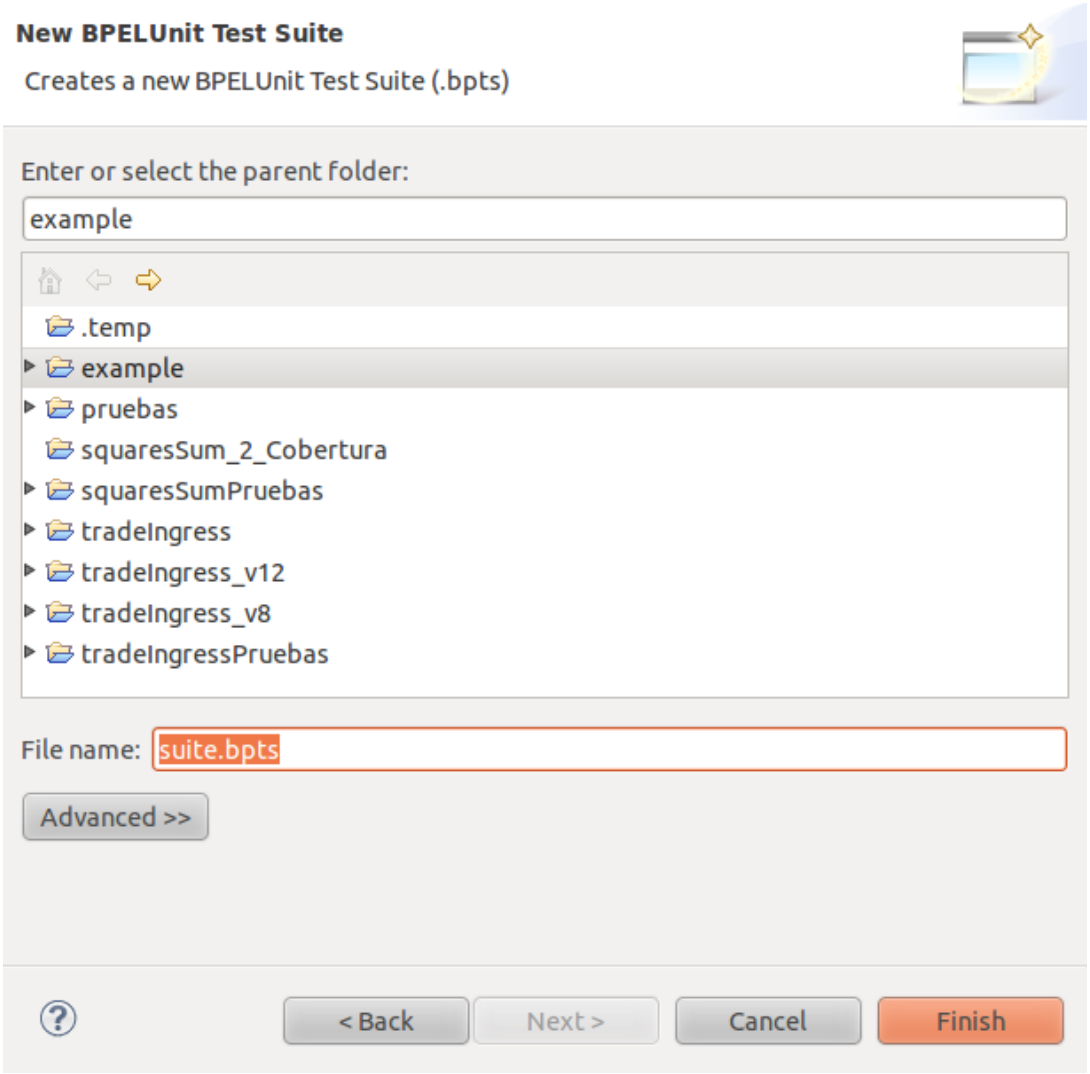

Figura A.13: Nombre fichero BPTS

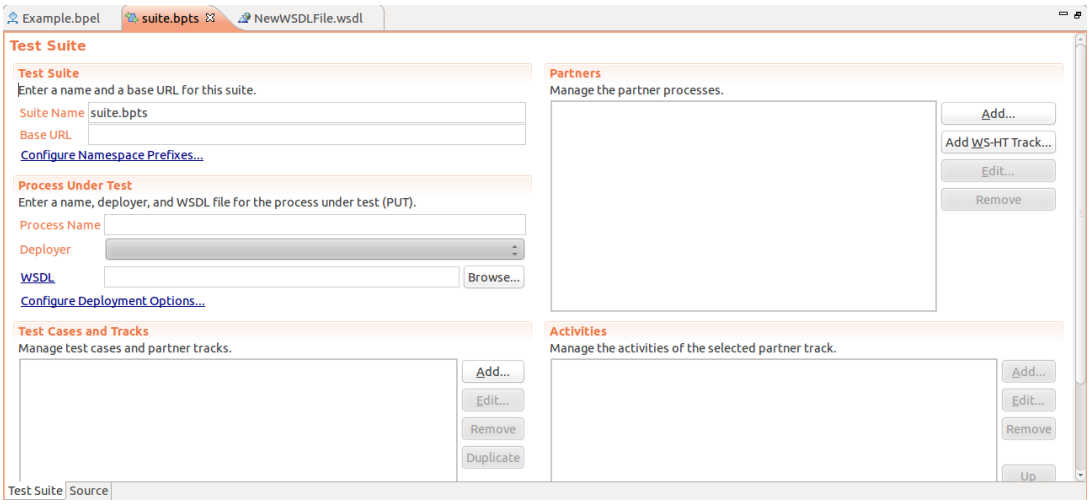

Figura A.14: Fichero BPTS

Apéndice B

Operadores de mutación para WS-BPEL 2.0

En esta sección del documento se detallan los operadores disponibles para WS-BPEL 2.0

Para la aplicación de la prueba de mutaciones es necesario disponer de un conjunto de operadores de mutación, que introduzcan cambios en el programa original mediante la aplicación de determinadas reglas sintácticas.

Los operadores definidos para WS-BPEL 2.0 se pueden dividir en cinco categorías, identificándose cada una con una letra mayúscula. Las categorías son las siguientes:

- Operadores de mutación de identificadores (I).
- Operadores de mutación de expresiones (E).
- Operadores de mutación de actividades (A).
- Operadores de mutación de condiciones excepcionales y eventos (X).
- Operadores de mutación de cobertura (C).

Cada operador de mutación definido para WS-BPEL 2.0 se define dentro una de estas cinco categorías y se identifican mediante tres letras mayúsculas: la primera de ellas hace referencia a la categoría a la que pertenece el operador, mientras que con las dos últimas letras se identifica a cada operador dentro de una determinada categoría.

Los operadores modelan los fallos que se pueden cometer al desarrollar una composición de servicios con WS-BPEL 2.0 y hacen que se apliquen ciertos criterios de cobertura sobre estos servicios. En la definición de los operadores se ha hecho especial hincapié en evitar que produzcan muchos mutantes equivalentes y también mutantes no válidos. Un mutante se dice que es no válido cuando no se puede ejecutar porque viola alguna de las restricciones estáticas de WS-BPEL 2.0.

B.1. Operadores de mutación de identificadores

Mediante estos operadores es posible modelar el cambio del identificador de una variable por el de otra. El operador de mutación definido en WS-BPEL encargado de esto es el operador ISV, que sustituye el identificador de una variable por el de otra del mismo tipo si ambas pertenecen al mismo ámbito. Al ser las variables del mismo tipo, se evita que se produzca un error en el intercambio de identificadores y se evita que se generen mutantes no válidos.

Veamos el siguiente ejemplo. En él, se invoca a dos WS de dos compañías de vuelo para realizar una reserva. Cada compañía devuelve su respuesta en una variable distinta, RespuestavueloAE y RespuestaVueloIb, respectivamente. Mediante el operador ISV se realiza el cambio de la variable respuestaVueloIb por la variable respuestaVueloAE.

Código original:

```
<flow>
  <sequence>
    <invoke partnerLink="AirEuropa" . . .
    inputVariable="DetalleVuelo" />
    <receive partnerLink="AirEuropa" . . .
    variable="RespuestaVueloAE"
  </sequence>
   <sequence>
    <invoke partnerLink="Iberia" . . .
    inputVariable="DetalleVuelo" />
    <receive partnerLink="Iberia" . . .
    variable="RespuestaVueloIb"
  </sequence>
</flow>
```

Mutante generado por ISV:

```
<flow>
  <sequence>
    <invoke partnerLink="AirEuropa" . . .
    inputVariable="DetalleVuelo" />
    <receive partnerLink="AirEuropa" . . .
    variable="RespuestaVueloAE"
  </sequence>
  <sequence>
    <invoke partnerLink="Iberia" . . .
    inputVariable="DetalleVuelo" />
    <receive partnerLink="Iberia" . . .
    variable="RespuestaVueloAE"
  </sequence>
</flow>
```

B.2. Operadores de mutación de expresiones

Mediante estos operadores es posible modelar el cambio, dentro de una expresión, de un operador por otro del mismo tipo. Los tipos de expresiones utilizados en el lenguaje WS-BPEL 2.0 pueden ser booleanas, de duración, de fecha límite, etc.

Los operadores de mutación de expresiones definidos para WS-BPEL 2.0 son los siguientes:

- **EAA:** Sustituye un operador aritmético (+, -, *, div, mod) por otro del mismo tipo en una expresión aritmética.
- **EEU:** Elimina el operador - unario de cualquier expresión.
- **ERR:** Sustituye un operador relacional (<, > , <=, >=, =, !=) por otro del mismo tipo en una expresión relacional.

- **ELL:** Sustituye un operador lógico (and, or) por otro del mismo tipo en una expresión lógica.
- **ECC:** Sustituye un operador de camino (/, //) por otro del mismo tipo en una expresión de camino.

  Veamos un ejemplo de como se aplicaría uno de los operadores de mutación anteriores:

  Código original:

```
$tienda.stock > 1000
```

  Mutante generado por ERR:

```
$tienda.stock < 1000
```

- **ECN:** Modifica las constantes numéricas de varias formas para modelar los posibles errores que se pueden cometer al introducirlas. Las modificaciones son:
  - Incrementar el valor de la constante numérica en una unidad.
  - Decrementar el valor de la constante numérica en una unidad.
  - Repetir el último número de la constante numérica.
  - Eliminar el último número de la constante numérica.

  Para evitar que se produzcan mutantes repetidos, se consideran dos casos especiales. En el caso de que la constante sea 0, las modificaciones resultantes serían 1, -1, 10 y -10. Y en el caso de que sea 1, las modificaciones resultantes serían 2, 0, 11 y -1.

  Veamos un ejemplo de como se aplicaría el operador de mutación ECN:

  Código original:

```
$tienda.stock > 1000
```

  Mutante generado por ECN:

```
$tienda.stock > 1001
```

- **EMD:** Este operador se ha definido para modificar las expresiones de duración. Lo hace de dos formas: sustituyendo la expresión de duración por 0, por lo que la condición se cumpliría inmediatamente, y por la mitad del valor inicial; para observar si el margen de seguridad especificado por la duración es adecuado.

  Veamos un ejemplo de como se aplicaría el operador de mutación EMD:

  Código original:

```
<onAlarm>
  <for>'P0DT8H'</for>
</onAlarm>
```

  Mutante generado por EMD:

```
<onAlarm>
  <for>'P0DT4H'</for>
</onAlarm>
```

- **EMF:** Mediante EMF se modifican las expresiones de fecha límite, sustituyendo la fecha inicial por la fecha del sistema.

  Veamos un ejemplo de como se aplicaría el operador de mutación EMF:

  Código original:

```
<onAlarm>
  <until>'2012-11-01'</until>
</onAlarm>
```

  Mutante generado por EMF:

```
<onAlarm>
  <until>'2012-06-01'</until>
</onAlarm>
```

Dentro de esta categoría también se incluyen unos operadores relacionados con los criterios de cobertura, ya que, aunque no cumplan ninguno de un modo específico, pueden aplicar ciertos criterios de cobertura dentro de un determinado contexto. Son los siguientes:

- **EIU:** Este operador inserta el menos unario en una expresión aritmética de primer nivel, es decir, si la expresión está formada por subexpresiones, no se aplicaría a estas, sólo a la expresión completa.

  Veamos un ejemplo de como se aplicaría el operador de mutación EIU:

  Código original:

```
<sequence>
  ...
  <assign>
    <copy>
      <from>$var1 * $var2</from>
      <to variable="result"/>
    </copy>
  </assign>
  ...
</sequence>
```

  Mutante generado por EIU:

```
<sequence>
  ...
  <assign>
    <copy>
      <from>-($var1 * $var2)</from>
      <to variable="result"/>
    </copy>
```

```
    </assign>
    ...
</sequence>
```

- **EIN:** Mediante este operador se inserta el operador de negación en una expresión lógica de primer nivel.

  Veamos un ejemplo de como se aplicaría el operador de mutación EIN:

  Código original:

```
<sequence>
  ...
  <if>
    <condition>$var1 > 10</condition>
    ...
  </if>
</sequence>
```

  Mutante generado por EIN:

```
<sequence>
  ...
  <if>
    <condition>not($var1 > 10)</condition>
    ...
  </if>
</sequence>
```

- **EAP:** Este operador inserta el valor absoluto positivo en una expresión aritmética de primer nivel.

  Veamos un ejemplo de como se aplicaría el operador de mutación EAP:

  Código original:

```
<sequence>
  ...
  <assign>
    <copy>
      <from>$var1 * $var2</from>
      <to variable="result"/>
    </copy>
  </assign>
  ...
</sequence>
```

  Mutante generado por EAP:

```
<sequence>
  ...
  <assign>
    <copy>
      <from>(($var1 * $var2) * (((($var1 * $var2) >= 0) - (($var1 * $
          var2) < 0)))</from>
```

```
      <to variable="result"/>
    </copy>
  </assign>
  ...
</sequence>
```

- **EAN:** Este operador inserta el valor absoluto negativo en una expresión aritmética de primer nivel.

  Veamos un ejemplo de como se aplicaría el operador de mutación EAN:

  Código original:

```
<sequence>
  ...
  <assign>
    <copy>
      <from>$var1 * $var2</from>
      <to variable="result"/>
    </copy>
  </assign>
  ...
</sequence>
```

  Mutante generado por EAN:

```
<sequence>
  ...
  <assign>
    <copy>
      <from>-(($var1 * $var2) * ((($var1 * $var2) >= 0) - (($var1 * $
          var2) < 0)))</from>
      <to variable="result"/>
    </copy>
  </assign>
  ...
</sequence>
```

B.3. Operadores de mutación de actividades

Los operadores de mutación incluidos en esta categoría actúan sobre las actividades que proporciona el lenguaje WS-BPEL 2.0.

Uno de los propósitos de estos operadores es eliminar una actividad o sustituirla por otra para poder modelar el fallo que se puede cometer si se elige una actividad que no es la más indicada para realizar una determinada acción. También se utilizan para modelar la elección de un valor erróneo para los atributos de las actividades, mediante la sustitución del valor actual por otro que sí sea correcto.

Los operadores de mutación de actividades se clasifican en dos tipos, los relacionados con la concurrencia y los no concurrentes:

B.3.1. Relacionados con la concurrencia

Dentro de este tipo de operadores se encuentran los siguientes operadores:

- **ACI:** Operador que cambia el atributo *createInstance* de una actividad de recepción de mensajes de "yes" a "no", siempre que haya más de una en la composición. Es decir, convierte una actividad de entrada al proceso en una actividad que no lo es.

  Además, a la hora de aplicar el operador ACI se tiene en cuenta una restricción para que no produzca mutantes no válidos. Teniéndose en cuenta los siguientes casos:

```
<process>
  <sequence>
    <flow>
      <receive createInstance="yes" /> → se descarta
      <receive createInstance="yes" /> → se descarta
    </flow>
  </sequence>
</process>


<process>
  <sequence>
    <flow>
      <receive createInstance="yes" /> → se descarta
        <sequence>
          <receive createInstance="yes" /> → se descarta
          ...
        </sequence>
    </flow>
  </sequence>
</process>


<process>
  <sequence>
    <receive createInstance="yes"/> → se descarta
    ...
    <flow>
      <receive createInstance="yes"/> → produce mutante 1
      <receive createInstance="yes"/> → produce mutante 2
    </flow>
    ...
  </sequence>
</process>
```

  En los dos primeros, ACI no produce ningún mutante debido a que serían no válidos según las comprobaciones estáticas que realiza WS-BPEL. Sin embargo, el tercer fragmento de código sí produciría dos mutantes ya que el flow en el que están los dos receive tiene a un sequence como padre sin estar el primero.

- **AFP:** Este operador tiene como objetivo cambiar el carácter de una actividad *forEach* de secuencial a paralelo, mediante la modificación del valor del atributo *parallel* de dicha actividad, cambiándolo de "no" a "yes". No se tiene en cuenta el cambio contrario del valor del atributo, de "yes" a "no", ya que no podría reflejar ningún error en los resultados del proceso al pasar la actividad *forEach* de carácter paralelo a carácter secuencial, por lo que se generaría un mutante equivalente.

  Veamos un ejemplo de como se aplicaría el operador de mutación AFP:

Código original:

```
<forEach parallel="no" . . . >
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>10</finalCounterValue>
  <scope>...</scope>
</forEach>
```

Mutante generado por EFP:

```
<forEach parallel="yes" . . . >
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>10</finalCounterValue>
  <scope>...</scope>
</forEach>
```

- **ASF:** Mediante este operador se realiza el cambio de una actividad *sequence* por una actividad *flow*. Al igual que con el operador ASF no se realiza el cambio contrario porque la estructura de una actividad *flow* es más compleja que la de una actividad *sequence*, para evitar la generación de demasiados mutantes equivalentes.

  Veamos un ejemplo de como se aplicaría el operador de mutación ASF:

  Código original:

```
  <sequence name="ReservaBillete">
    <invoke partnerLink="Iberia" . . ./>
    <receive partnerLink="Iberia" . . ./>
  </sequence>
```

  Mutante generado por ASF:

```
  <flow name="ReservaBillete">
    <invoke partnerLink="Iberia" . . ./>
    <receive partnerLink="Iberia" . . ./>
  </flow>
```

- **AIS:** Este operador cambia el atributo *isolated* de “yes” a “no” en un *scope* en el que se manipulan variables compartidas, No protegiéndose de esta manera el acceso a estas variables, por lo que las actividades de otro *scope* podrían acceder a ellas a la vez, produciéndose resultados impredecibles. En este operador no se tiene en cuenta, tampoco, el cambio contrario puesto que generaría mutantes equivalentes.

  Veamos un ejemplo de como se aplicaría el operador de mutación AIS:

  Código original:

```
<flow>
  <scope name="Scope1" isolated="yes">
    <sequence>
      <assign>
  <copy>
    <from> . . . </from>
```

```
7       <to variable="variableGlobal" />
8     </copy>
9         </assign>
10        ...
11      </sequence>
12    </scope>
13    <scope name="Scope2" isolated="yes" >
14      <sequence>
15        <invoke outputVariable="variableGlobal" .../>
16        ...
17      </sequence>
18    </scope>
19  </flow>
```

Mutante generado por AIS:

```
1   <flow>
2     <scope name="Scope1" isolated="no">
3       <sequence>
4         <assign>
5     <copy>
6       <from> . . . </from>
7       <to variable="variableGlobal" />
8     </copy>
9         </assign>
10        ...
11      </sequence>
12    </scope>
13    <scope name="Scope2" isolated="yes" >
14      <sequence>
15        <invoke outputVariable="variableGlobal" .../>
16        ...
17      </sequence>
18    </scope>
19  </flow>
```

B.3.2. No concurrentes

Los operadores de mutación incluidos en este tipo son los siguientes:

- **AEL:** Mediante el operador AEL se elimina una actividad, excepto la actividad *receive* o *pick* inicial.

  Veamos un ejemplo de como se aplicaría el operador de mutación AEL:

  Código original:

```
1     <sequence name="ReservaBillete">
2       <invoke partnerLink="Iberia" . . ./>
3       <receive partnerLink="Iberia" . . ./>
4     </sequence>
```

  Mutante generado por AEL:

```
  <sequence name="ReservaBillete">

    <receive partnerLink="Iberia" . . ./>
  </sequence>
```

- **AIE:** Con el operador AIE se puede modelar el olvido de una rama de una actividad *if* mediante la eliminación de un elemento *elseif* o del elemento *else*.

  Veamos un ejemplo de como se aplicaría el operador de mutación AIE:

  Código original:

```
<if name="StockMenor1000">
  <condition>$stock &lt; 1000</condition>
  <invoke name="Reserva300Items" . . . />
  <elseif>
    <condition>$pedido &lt; 500</condition>
    <invoke name="Reserva300Items" . . . />
  </elseif>
  <else>
    <reply name="SinReserva" . . . />
  </else>
</if>
```

  Mutante generado por AIE:

```
<if name="StockMenor1000">
  <condition>$stock &lt; 1000</condition>
  <invoke name="Reserva300Items" . . . />



  <else>
    <reply name="SinReserva" . . . />
  </else>
</if>
```

- **AWR:** El operador AWR modela el error que podría cometerse al no seleccionar el tipo de actividad repetitiva adecuada. Para ello, cambia una actividad *while* por otra *repeatUntil* y viceversa.

  Veamos un ejemplo de como se aplicaría el operador de mutación AWR:

  Código original:

```
<while>

  <condition>
    $iteraciones &lt; 10
  </condition>
  <scope name="calculateProfits">...</scope>
</while>
```

  Mutante generado por AWR:

```
<reapeatUntil>
  <scope name="calculateProfits">...</scope>
  <condition>
    $iteraciones &lt; 10
  </condition>

</reapeatUntil>
```

- **AJC:** Mediante el operador AJC se elimina el elemento *joinCondition* de un determinado contenedor *targets*, para simular el error que se cometería al no incluir una condición distinta de la disyunción del estado de todos los *target*.

  Veamos un ejemplo de como se aplicaría el operador de mutación AJC:

  Código original:

```
<targets>
  <target linkName="link1"/>
  <target linkName="link2"/>
  <joinCondition>$link1 and $link2</joinCondition>
</targets>
```

  Mutante generado por AJC:

```
<targets>
  <target linkName="link1"/>
  <target linkName="link2"/>

</targets>
```

- **ASI:** Con el operador ASI se intercambia el orden de dos actividades que sean hijas de una actividad *sequence*. Cuando las actividades que se intercambian no tienen una dependencia de datos entre ellas, se producen mutantes equivalentes.

  Veamos un ejemplo de como se aplicaría el operador de mutación ASI:

  Código original:

```
  <sequence name="ReservaBillete">
    <invoke partnerLink="Iberia" . . ./>
    <receive partnerLink="Iberia" . . ./>
  </sequence>
```

  Mutante generado por ASI:

```
  <sequence name="ReservaBillete">
    <receive partnerLink="Iberia" . . ./>
    <invoke partnerLink="Iberia" . . ./>
  </sequence>
```

- **APM:** El operador APM elimina un evento *onMessage*, siempre que exista más de uno, de una actividad *pick*, para modelar el error que podría cometerse al olvidar la posible recepción de un mensaje.

Veamos un ejemplo de como se aplicaría el operador de mutación APM:

Código original:

```
<pick>
  <onMessage partnerLink="cliente"
    operation="reservarArticulo" . . . > . . .
  </onMessage>
  <onMessage partnerLink="cliente"
    operation="pedidoCompleto" . . . > . . .
  </onMessage>
  <onAlarm>
    <for>'P2DT12H'</for> . . .
  </onAlarm>
</pick>
```

Mutante generado por APM:

```
<pick>
  <onMessage partnerLink="cliente"
    operation="reservarArticulo" . . . > . . .
  </onMessage>


  <onAlarm>
    <for>'P2DT12H'</for> . . .
  </onAlarm>
</pick>
```

- **APA:** El operador APA elimina el elemento *onAlarm* que puede aparecer en una actividad *pick* o en un manejador de eventos, para modelar el error que se podría cometer al olvidar introducir un elemento *onAlarm*.

  Veamos un ejemplo de como se aplicaría el operador de mutación APA:

  Código original:

```
<pick>
  <onMessage partnerLink="cliente"
    operation="reservarArticulo" . . . > . . .
  </onMessage>
  <onMessage partnerLink="cliente"
    operation="pedidoCompleto" . . . > . . .
  </onMessage>
  <onAlarm>
    <for>'P2DT12H'</for> . . .
  </onAlarm>
</pick>
```

  Mutante generado por APM:

```
<pick>
  <onMessage partnerLink="cliente"
    operation="reservarArticulo" . . . > . . .
```

```
    </onMessage>
    <onMessage partnerLink="cliente"
      operation="pedidoCompleto" . . . > . . .
    </onMessage>


</pick>
```

B.4. Operadores de mutación de condiciones excepcionales y eventos

En esta categoría están incluidos los operadores de mutación relacionados con los distintos tipos de manejadores proporcionados por el lenguaje WS-BPEL: de fallos, de eventos, de compensación y de terminación.

- **XMF:** Con el operador XMF se modela el olvido de insertar un manejador específico para un fallo determinado o para cualquier fallo no especificado anteriormente, eliminando para ello, un elemento *catch* o el elemento *catchall* de un manejador de fallos, respectivamente.

  Veamos un ejemplo de como se aplicaría el operador de mutación XMF:

  Código original:

```
<faultHandlers>
  <catch faultName="SinExistencias" . . . >
    ...
  </catch>
  <catchAll> . . . </catchAll>
</faultHandlers>
```

  Mutante generado por XMF:

```
<faultHandlers>
  <catch faultName="LibroNoDisponible" . . . >
    ...
  </catch>

</faultHandlers>
```

- **XMC:** Mediante el operador XMC se elimina la definición de un manejador de compensación. Sustituyéndolo, WS-BPEL, por el manejador de compensación por omisión.

  Veamos un ejemplo de como se aplicaría el operador de mutación XMC:

  Código original:

```
<scope name="compra" . . . >
  ...
  <faultHandlers>
    <catch faultName="SinExistencias" . . . >
      ...
    </catch>
    <catchAll> . . . </catchAll>
  </faultHandlers>
```

```
  <compensationHandler>
    <invoke partnerLink="Vendedor" . . . />
  </compensationHandler>
</scope>
```

Mutante generado por XMC:

```
<scope name="compra" . . . >
  ...
  <faultHandlers>
    <catch faultName="SinExistencias" . . . >
      ...
    </catch>
    <catchAll> . . . </catchAll>
  </faultHandlers>


</scope>
```

- **XMT:** Mediante el operador XMT se elimina la definición de un manejador de terminación. Sustituyéndolo, WS-BPEL, por el manejador de terminación por omisión.

  Veamos un ejemplo de como se aplicaría el operador de mutación XMT:

  Código original:

```
<scope name="compra" . . . >
  ...
  <faultHandlers>
    <catch faultName="SinExistencias" . . . >
      ...
    </catch>
    <catchAll> . . . </catchAll>
  </faultHandlers>
  <terminationHandler>
    <invoke partnerLink="Vendedor" . . . />
  </terminationHandler>
</scope>
```

  Mutante generado por XMT:

```
<scope name="compra" . . . >
  ...
  <faultHandlers>
    <catch faultName="SinExistencias" . . . >
      ...
    </catch>
    <catchAll> . . . </catchAll>
  </faultHandlers>


</scope>
```

- **XTF:** Con el operador XTF se sustituye el nombre, indicado en el atributo *faultName*, del fallo lanzado por una actividad *throw* por el nombre de otro fallo, siempre que ambos sean del mismo ámbito. Modelando, así, la elección errónea a la hora de lanzar un fallo.

  Veamos un ejemplo de como se aplicaría el operador de mutación XTF:

  Código original:

```
<if>
  <condition>$stock &gt;= $pedido</condition>
  <sequence> . . . </sequence>
  <elseif>
    <condition>$stock = 0</condition>
    <throw faultName="ItemNoDisp" />
  </elseif>
  <else>
    <throw faultName="NoHayStock" />
  </else>
</if>
```

  Mutante generado por XTF:

```
<if>
  <condition>$stock &gt;= $pedido</condition>
  <sequence> . . . </sequence>
  <elseif>
    <condition>$stock = 0</condition>
    <throw faultName="NoHayStock" />
  </elseif>
  <else>
    <throw faultName="NoHayStock" />
  </else>
</if>
```

- **XER:** El operador XER modela el olvido de la introducción de una actividad *rethrow* en un manejador de fallos, mediante la eliminación de éste.

  Veamos un ejemplo de como se aplicaría el operador de mutación XER:

  Código original:

```
<faultHandlers>
  <catch faultName="SinExistencias" . . . >
    ...
    <rethrow>...</rethrow>
  </catch>
  <catchAll> . . . </catchAll>
</faultHandlers>
```

  Mutante generado por XER:

```
<faultHandlers>
  <catch faultName="SinExistencias" . . . >
    ...

```

```
    </catch>
    <catchAll> . . . </catchAll>
</faultHandlers>
```

- **XEE:** El operador XEE modela el olvido a la hora de especificar un evento que puede recibir el proceso, mediante la eliminación de un elemento *onEvent* de un manejador de eventos.

  Veamos un ejemplo de como se aplicaría el operador de mutación XEE:

  Código original:

```
<eventHandlers>
  <onEvent partnerLink="cliente"
    operation="estadoPedido" . . . >
    <scope>. . .</scope>
  </onEvent>
  <onEvent partnerLink="cliente"
    operation="cancelarPedido" . . . >
    <scope>. . .</scope>
  </onEvent>
</eventHandlers>
```

  Mutante generado por XEE:

```
<eventHandlers>



  <onEvent partnerLink="cliente"
    operation="cancelarPedido" . . . >
    <scope>. . .</scope>
  </onEvent>
</eventHandlers>
```

B.5. Operadores de mutación de cobertura

Mediante estos operadores se consigue la aplicación de unos determinados criterios de cobertura sobre los servicios web. Los operadores de mutación de cobertura existentes son los siguientes:

- **CFA:** Mediante este operador se sustituye una actividad por la actividad *exit*. Con este operador se cubre el criterio de cobertura de sentencias, con el que se intenta conseguir la ejecución de todas las sentencias que compone el programa.

  Veamos un ejemplo de como se aplicaría el operador de mutación CFA:

  Código original:

```
<sequence>
  <assign>...</assign>
</sequence>
```

  Mutante generado por CFA:

```
<sequence>
  <exit/>
</sequence>
```

- **CDE:** El operador CDE sustituye una decisión por *true()* o *false()*. Con este operador se cubre el criterio de cobertura de decisión, con el que se intenta asegurar que se ejecuten todas las ramas del flujo del programa.

  Veamos un ejemplo de como se aplicaría el operador de mutación CDE:

  Código original:

```
<sequence>
  ...
  <if>
    <condition>
      ($var1 = 0 and $var2 > 10)
    </condition>
    ...
  </if>
</sequence>
```

  Mutante generado por CDE:

```
<sequence>
  ...
  <if>
    <condition>
      (true())
    </condition>
    ...
  </if>
</sequence>
```

- **CCO:** Mediante este operador se sustituye una condición por *true()* o *false()*. Con este operador se cubre el criterio de cobertura de condición, mediante el que se procura que dentro de cada rama o decisión, cada condición tome todos los valores posibles al menos una vez.

  Veamos un ejemplo de como se aplicaría el operador de mutación CCO:

  Código original:

```
<sequence>
  ...
  <if>
    <condition>
      ($var1 = 0 and $var2 > 10)
    </condition>
    ...
  </if>
</sequence>
```

  Mutante generado por CCO:

```
<sequence>
  ...
  <if>
    <condition>
      (true() and $var2 > 10)
    </condition>
    ...
  </if>
</sequence>
```

- **CDC:** El operador CDC sustituye una decisión o condición por *true()* o *false()*. Con este operador se cubre a la vez, tanto el criterio de cobertura de decisión, como el de condición. Por lo que, al aplicar este operador, se producirían los mismos mutantes que al aplicar el operador CDE y CCO.

En la siguiente tabla podemos encontrar un resumen de todos los operadores de mutación disponibles para WS-BPEL 2.0:

Operador	Descripción
	Mutación de identificadores
ISV	Sustituye el identificador de una variable por el de otra del mismo tipo
	Mutación de expresiones
EAA	Sustituye un operador aritmético (+, -, *, div, mod) por otro del mismo tipo
EEU	Elimina el operador - unario de cualquier expresión
ERR	Sustituye un operador relacional (=, !=, <, > , <=, >=) por otro del mismo tipo
ELL	Sustituye un operador lógico (and, or) por otro del mismo tipo
ECC	Sustituye un operador de camino (/, //) por otro del mismo tipo
ECN	Modifica una constante numérica incrementándola/decrementándola en una unidad, o añadiendo/eliminando un dígito
EMD	Modifica una expresión de duración, reemplazándola por 0 o por la mitad de su valor inicial
EMF	Modifica una expresión de fecha límite, reemplazándola por la fecha actual del sistema
	Mutación de expresiones relacionados con la cobertura
EIU	Inserta el menos unario en una expresión aritmética
EIN	Inserta la negación en una expresión lógica
EAP	Inserta el valor absoluto positivo en una expresión aritmética
EAN	Inserta el valor absoluto negativo en una expresión aritmética
	Mutación de actividades concurrentes
ACI	Cambia el atributo *createInstance* de una actividad de recepción de mensajes de *yes* a *no*
AFP	Sustituye una actividad *forEach* secuencial por una paralela
ASF	Sustituye una actividad *sequence* por una actividad *flow*
AIS	Cambia el atributo *isolated* de un scope de *yes* a *no*
	Mutación de actividades no concurrentes
AEL	Borra una actividad
AIE	Borra un elemento *elseif* o el elemento *else* de una actividad *if*
AWR	Sustituye una actividad *while* por una actividad *repeatUntil* y viceversa
AJC	Borra el atributo *joinCondition* de una actividad
ASI	Intercambia el orden de dos actividades hijas de una actividad *sequence*
APM	Borra un elemento *onMessage* de una actividad *pick*
APA	Borra una elemento *onAlarm* de una actividad *pick* o de un manejador de eventos
	Mutación de condiciones excepcionales y eventos
XMF	Borra un elemento *catch* o el elemento *catchAll* de un manejador de fallos
XMC	Borra la definición de un manejador de compensación
XMT	Borra la definición de un manejador de terminación
XTF	Sustituye el fallo lanzado por una actividad *throw*
XER	Borra una actividad *rethrow*
XEE	Borra un elemento *onEvent* de un manejador de eventos
	Mutación de cobertura
CFA	Sustituye una actividad por la actividad *exit*
CDE	Sustituye una decisión por *true()* o *false()*
CCO	Sustituye una condición por *true()* o *false()*
CDC	Sustituye una decisión o condición por *true()* o *false()*

Tabla B.1: Operadores de mutación para WS-BPEL 2.0

Apéndice C

Manual de usuario de MuBPEL

En esta sección describiremos las distintas funciones que proporciona la herramienta MuBPEL.

C.1. Listar operadores aplicables a una composición

Para obtener una lista con todos los operadores que se pueden aplicar a una determinada composición basta con aplicar la siguiente orden:

```
mubpel analyze fichero.bpel
```

En la lista generada, se obtiene el nombre de cada operador seguido de dos números: el primer número representa el número de localizaciones en las que se puede aplicar dicho operador en la composición y el segundo número representa el valor máximo del atributo de cada operador, es decir, el número de formas distintas en las que se puede aplicar en cada localización.

El número total de mutantes que se pueden generar de una composición se obtiene mediante la suma del número de mutantes que se pueden generar de cada operador. A su vez, el número total de mutantes que se pueden generar para cada operador se obtiene al multiplicar el número de localizaciones en las que se puede aplicar en la composición un determinado operador por el valor máximo del atributo de ese operador.

C.2. Generar mutantes de una composición

Mediante la siguiente orden podemos generar todos los mutantes que se pueden aplicar a una determinada composición.

```
mubpel applyall fichero.bpel
```

También tenemos la posibilidad de generar un mutante específico mediante la siguiente orden.

```
mubpel apply fichero.bpel operator operandIndex attribute > mutante.bpel
```

Donde *operator*, *operandIndex* y *attribute* son números que representan, respectivamente, al operador a aplicar, la localización en la composición donde aplicar el operador y la forma concreta en la que se debe aplicar.

C.3. Ejecución de una composición

Para ejecutar una composición debemos escribir la siguiente orden en una terminal. Esta orden nos permite observar los resultados obtenidos por la composición al ejecutarla contra el fichero de pruebas *bpts*. Para poder observar fácilmente los resultados obtenidos es conveniente almacenarlos en un fichero.

```
mubpel run fichero.bpts fichero.bpel > salida.xml
```

C.4. Comparar la salida de la composición y la de los mutantes

Podemos comparar la salida de la ejecución de la composición con la de los mutantes a través de dos órdenes, *compare* y *comparefull*

```
mubpel compare fichero.bpts fichero.bpel salida.xml (fichero1.bpel...|-)
```

Mediante *compare* comparamos la salida de la composición y la de los mutantes hasta que se produzca la primera diferencia entre ellas.

La salida que se obtiene consta de una fila por cada mutante con la que se realiza la comparación, dicha fila está formada por el mutante seguido de una serie de números, representando cada número al resultado obtenido para cada caso de prueba.

- Si el número es 0, las salidas de la composición y el mutante coinciden.
- Si el número es 1, las salidas de la composición y el mutante son distintas.
- Si el número es 2, se trata de un mutante inválido que no puede ser ejecutado.

Debido a que se compara hasta encontrar la primera diferencia, la comparación se detendrá al obtener el primer 1 de la fila y el resto de casos de prueba quedará a 0. Si se trata de un mutante inválido, la fila entera quedará a 2.

Por otro lado, podemos comparar la salida de la ejecución de la composición con la de los mutantes, para todos los casos de pruebas existentes, a través de *comparefull*. A diferencia de *compare*, se realiza la comparación teniendo en cuenta todos los casos de pruebas existentes, en lugar de detener la comparación al obtener la primera diferencia.

```
mubpel comparefull fichero.bpts fichero.bpel salida.xml (fichero1.bpel...|-)
```

C.5. Comparar dos salidas de ejecución de una composición

Podemos comparar dos salidas de la ejecución de una composición mediante la siguiente orden:

```
mubpel compareout salida1.xml salida2.xml
```

De esta manera, podemos observar si para cada caso de prueba se ha obtenido la misma salida.

C.6. Normalizar una composición

Podemos normalizar una composición WS-BPEL, es decir, generar una forma canónica de tal composición, mediante la orden:

```
mubpel normaliza fichero.bpel
```

Resulta muy útil para utilizar una herramienta para poder ver las posibles diferencias entre la composición y un mutante generado a partir de ella. Por lo que es recomendable redirigir la salida estándar a un fichero para poder realizar tal comprobación.

Bibliografía

[1] A. Estero-Botaro, F. Palomo-Lozano y I. Medina-Bulo. Mutation operators for WS-BPEL 2.0. En *ICSSEA 2008, 21th International Conference on Software & Systems Engineering and their applications*. 2008.

[2] A. Estero-Botaro, J. Boubeta-Puig, V. Liñeiro-Barea y I. Medina-Bulo. Operadores de mutación de cobertura para WS-BPEL 2.0. En *Actas de las XVI Jornadas de Ingeniería del Software y Bases de Datos*. 2012.

[3] A. García-Domínguez, A. Estero-Botaro, F. Palomo-Lozano y I. Medina-Bulo. MuBPEL: una herramienta de mutación firme para WS-BPEL 2.0. En *Actas de las XVI Jornadas de Ingeniería del Software y Bases de Datos*. 2012.

[4] A. J. Offutt y R. H. Untch. *Mutation Testing for the New Century*, capítulo Mutation 2000: Uniting the Orthogonal, páginas 34–44. Kluwer Academic Publishers, 2001.

[5] OASIS. Web Services Business Process Execution Language 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, 2007.

[6] W3C. Web Services Description Language 2.0. `http://www.w3.org/TR/wsdl20`, Junio 2007.

[7] W3C. Simple Object Access Protocol. `http://www.w3.org/TR/soap12-part1`, Abril 2007.

[8] W3C. Xml schema. `http://www.w3.org/TR/xmlschema-2`, Octubre 2004.

[9] W3C. Xpath. `http://www.w3.org/TR/xpath20`, Diciembre 2010.

[10] BPELUnit. The open source unit testing framework for bpel. `http://bpelunit.net/`, Septiembre 2011.

[11] Apache. Velocity. `http://velocity.apache.org/`, Noviembre 2010.

[12] M. Ángel Pérez Montero. TestSpec. En *Generador de casos de prueba aleatorio basado en especificaciones abstractas*. 2012.

[13] LaTeX Project Team. LaTeX - a document preparation system. `http://www.latex-project.org/`, 2010.

[14] L. Lamport. *LaTeX - A Document Preparation System: User's Guide and Reference Manual, Second Edition*. Pearson / Prentice Hall, 1994.

[15] sourceforge.net. TkDiff. `http://sourceforge.net/projects/tkdiff/`, Mayo 2012.

[16] A. G. Domínguez. XMLEye. `https://neptuno.uca.es/redmine/projects/sources-fm/wiki/XMLEye`, Septiembre 2011.

[17] ActiveBPel. `https://neptuno.uca.es/redmine/projects/activebpel`, Septiembre 2011.

[18] T. E. Foundation. ECLIPSE. `https://www.eclipse.org/`, Septiembre 2011.

[19] The Web Services-Interoperability Organization. Web Services Interoperability Basic Profile. `http://www.ws-i.org/Profiles/BasicProfile-1.1.html`, Febrero 2006.

[20] B. Collins-Sussman, B. W. Fitzpatrick y C. M. Pilato. Version control with subversion. `http://svnbook.red-bean.com/en/1.2/index.html`, 2006.